

**LARGE-SCALE SGD ALGORITHMS
AND
THE EXPRESSIVE POWER OF MODERN NEURAL NETWORKS**

by
Shashank Rajput

A dissertation submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy
(Computer Sciences)

at the
UNIVERSITY OF WISCONSIN–MADISON
2023

Date of final oral examination: April 13th, 2023

The dissertation is approved by the following members of the Final Oral Committee:

Dimitris Papailiopoulos, Associate Professor, Electrical & Computer Engineering
Stephen J. Wright, Professor, Computer Sciences
Kangwook Lee, Assistant Professor, Electrical & Computer Engineering
Xiaojin (Jerry) Zhu, Professor, Computer Sciences

© Copyright by Shashank Rajput 2023

All Rights Reserved

Dedicated to my loving family.

ACKNOWLEDGMENTS

I owe a great debt of gratitude to my doctoral advisor, Prof. Dimitris Papailiopoulos, whose help and guidance have been invaluable throughout my graduate career. Working in his lab has been an incredibly productive and enjoyable experience, during which I have gained numerous research and professional skills. Prof. Papailiopoulos has also been a skilled mentor, advising me on various academic, research, and career related topics; as well as coaching me over multiple years on how to give talks with confidence and ease, and in the process, also helping me to overcome my fear of public speaking. Working with him, I was also able to hone my ability to generate novel research ideas and critically evaluate them. Furthermore, he has consistently provided me with thoughtful and practical advice, particularly during the initial phase of my PhD, when I was struggling to balance academic demands and the work-life balance challenges posed by the COVID-19 pandemic. I feel incredibly fortunate to have had such an exceptional advisor and look forward to continuing to collaborate with him on future research projects.

I also thank Prof. Kangwook Lee for the many research discussions and collaborations. I am inspired his ability to draw insightful conclusions from research discussions and paper presentations, which is an ability I have tried to learn from him. I would also like to express my deep appreciation to Prof. Stephen J. Wright, Prof. Xiaojin (Jerry) Zhu, Prof. Robert D. Nowak, and Prof. Varun Jog for their outstanding teaching on optimization, machine learning, and information theory. The courses they taught have equipped me with the essential mathematical and theoretical knowledge and tools necessary for my research. I am grateful to Prof. Wright, Prof. Nowak, and others for organizing IFDS and SILO, each of which have been great venues to learn about research from within and outside the university, as well as serving as great platforms for initiating discussions and collaborations. I also thank Prof. Papailiopoulos, Prof. Wright, Prof. Lee, and Prof. Zhu for serving on my doctoral defense committee.

During my research internship at Google Brain, I had the privilege of working with

Dr. Maheswaran Sathiamoorthy and Dr. Raghunandan Keshawan. I feel fortunate to have had their considerate and supportive guidance as I familiarized myself with the field of recommender systems, which was relatively new to me, and as I learnt Google’s internal frameworks. I would like to extend my gratitude to Dr. Lichan Hong and Dr. Ed Chi for their valuable suggestions on my project and for the excellent advice they gave me during our meetings. Our discussions also provided me with insights into the subtle differences between academic and industrial research, as I gained experience in industrial research.

I am also grateful to Prof. Suvrit Sra for hosting me during Summer 2021 during my (remote) internship at MIT. I also thank Prof. Chulhee Yun, from whom I learnt a lot while collaborating during the internship. Although I was worried initially about the internship because it was remote (due to the COVID pandemic), working with Prof. Sra and Prof. Yun turned out to be a very productive and enjoyable experience, which I greatly cherish.

I am also grateful to Google for awarding me the Google Fellowship and Dr. H. Brendan McMahan for being my mentor for the fellowship. I also thank IFDS for awarding me the IFDS Research Assistant Fellowship for two years; and MADlab for supporting my doctoral research. ¹

I was also very fortunate to have had collaborated with many excellent researchers and wonderful people during my graduate career. Some of them include Prof. Stephen J. Wright, Prof. Amin Karbasi, Prof. Suvrit Sra, Prof. Jy-yong Sohn, Prof. Chulhee Yun, Dr. Hongyi Wang, Dr. Zachary Charles, Angeliki Giannou, Ankit Pensia, Kartik Sreenivas, Nikhil Mehta, Tuan Dinh, Alliot Nagle, Nayoung Lee, Zhili Feng, and Anant Gupta. I learnt very diverse research and professional viewpoints from each of these people. I thank Dr. Hongyi Wang and Dr. Zachary Charles not only providing great advice and guidance during my first few research projects, but also being great friends. I am also grateful to Ankit Pensia, from whom I have learnt not only a lot of mathematics, but also a lot of work

¹The research in this thesis was supported by an NSF CAREER Award #1844951, a Sony Faculty Innovation Award, an AFOSR & AFRL Center of Excellence Award FA9550-18-1-0166, an NSF TRIPODS Award #1740707, and an ONR Grant N00014-21-1-2806.

ethics.

I thank all of my close friends that I have made in Madison: Ankit Pensia, Muni Sreenivas Pydi, Vishnu Lokhande, Ashish Hooda, Rishabh Khandelwal, Anant Gupta, Aditya Kumar Akash, Shantanu Gupta, and Amanpreet Singh Saini for making this 6 year-long graduate journey fun and memorable. Ankit has been a great friend and roommate, and I thank him for all the joys and laughs, supporting me during times of uncertainty, and helping me evolve (for better or worse :D) over the past 4 years. I also thank my other roommate, Muni for our witty banter, deep philosophical discussions, and providing me alternative viewpoints on many topics. I thank Vishnu for being an awesome roommate during our summer internship in California; Ashish, Rishabh, and Shantanu for the great board games nights; and Anant, Aditya, and Amanpreet for the amazing road trips and dinner parties.

I will forever be grateful to my parents, Geeta Rani and Rakesh Kumar, and my sister Sfurti Rajput (Mannu didi :D) for fully encouraging and supporting all the life decisions that have led to my doctoral degree, and for being my support system that I know I can always count on.

ABSTRACT

This thesis focuses on two important aspects of modern Machine Learning (ML) that have significantly contributed to its success: the fast convergence of Shuffling-Based Stochastic Gradient Descent, and the large expressive power of modern neural architectures. While the latter is necessary for models to perform well at complex tasks, the the former has enabled their time- and computation-efficient training.

Shuffling-Based SGD, also known as Random Reshuffling (abbreviated as *RR*) or SGD Without-Replacement, refers to the most common variant of Stochastic Gradient Descent used in practice, where the sampling of data points at each iteration is done without replacement. While it has been known empirically for over a decade that *RR* leads to a faster convergence than vanilla SGD (which does with-replacement sampling), its theoretical understanding has lagged until recently. In this thesis, we provide tight theoretical bounds on the convergence rates of *RR* in both centralized and distributed (Federated Learning) setups. We also prove optimality results for the randomness used in shuffling, and provide a new algorithm, *FLIPFLOP*, which speeds up the convergence of *RR* on quadratic functions.

The second half of this thesis analyzes the expressive power of modern neural architectures. First we prove that the Transformer architecture is a universal computer which is capable of not just executing instructions, but also performing function approximation and complex linear algebra operations. We also show that it can simulate backpropagation over another network, hence giving a possible explanation for the emergent phenomenon of In-Context Learning in Large Language Models. Then, motivated by the empirical success of fine-tuning just the normalization layers (*e.g.*, BatchNorm layers) of large pre-trained networks, we theoretically prove that such layers indeed have surprising approximation power. Next, we prove the Strong Lottery Ticket Hypothesis, which says that just pruning the weights of a randomly initialized network is sufficient to approximate any (logarithmically smaller) target network. Finally, we improve exponentially the bounds on the memorization power of Multi-Layered Perceptrons.

CONTENTS

Abstract	v
Contents	vi
List of Tables	xii
List of Figures	xiii
List of Algorithms	xviii
I Introduction	1
1 Motivation	2
1.1 List of Papers	3
2 Shuffling-Based Stochastic Gradient Descent	5
2.1 Preliminaries	5
2.2 Summary of Results	9
2.3 Methodology	15
3 Expressive Power of Modern Neural Architectures	20
3.1 Preliminaries	20
3.2 Results and Proof Techniques	25
II Shuffling-Based SGD (SGD Without Replacement)	36
4 Tight Convergence Bounds for Random Reshuffling	37
4.1 Introduction	37
4.2 Related Work	41

4.3	<i>Preliminaries and Notation</i>	41
4.4	<i>Optimal RR Convergence Rates for Quadratics</i>	42
4.5	<i>Lower Bound for General Case</i>	50
4.6	<i>Conclusions and Future Work</i>	58
5	Random Reshuffling in Distributed and Federated Learning	60
5.1	<i>Introduction</i>	60
5.2	<i>Problem setup</i>	63
5.3	<i>Convergence analysis of minibatch and local RR</i>	68
5.4	<i>Matching lower bounds</i>	72
5.5	<i>Synchronized shuffling: how to bypass lower bounds</i>	76
5.6	<i>Conclusion</i>	79
6	Random Reshuffling: Is Random Optimal?	80
6.1	<i>Introduction</i>	80
6.2	<i>Related Work</i>	84
6.3	<i>Preliminaries</i>	86
6.4	<i>Exponential Convergence in 1-Dimension</i>	87
6.5	<i>Lower Bounds for Shuffling-based SGD</i>	89
6.6	<i>Flipping Permutations for Faster Convergence in Quadratics</i>	91
6.7	<i>Conclusion and Future Work</i>	97
7	Subsequent Research	98
	III The Expressive Power of Modern Neural Architectures	99
8	Looped Transformers as Programmable Computers	100
8.1	<i>Introduction</i>	100
8.2	<i>Prior Work</i>	105

8.3	<i>Preliminaries</i>	107
8.4	<i>Building Transformer Blocks towards General Computation</i>	111
8.5	<i>Emulating a Generalized One-instruction Set Computer</i>	121
8.6	<i>Functions in the Unified Template Form</i>	136
8.7	<i>A Basic Calculator</i>	141
8.8	<i>Linear Algebra</i>	145
8.9	<i>Emulating Learning Algorithms at Inference Time</i>	149
8.10	<i>Conclusion and Open Problems</i>	156
9	Expressive Power of Normalization Layers	158
9.1	<i>Introduction</i>	158
9.2	<i>Preliminaries</i>	164
9.3	<i>Main Results</i>	166
9.4	<i>Our Techniques</i>	169
9.5	<i>Discussion and Future Work</i>	177
10	Logarithmic Bounds for Strong LTH via the Subset Sum Problem	179
10.1	<i>Introduction</i>	179
10.2	<i>Preliminaries and Setup</i>	185
10.3	<i>Lottery Tickets via Subset Sum</i>	186
10.4	<i>Lower Bound by Parameter Counting</i>	193
10.5	<i>Experiments</i>	194
10.6	<i>Discussion</i>	197
11	Logarithmic Bounds for Strong LTH for Binary Networks	198
11.1	<i>Introduction</i>	198
11.2	<i>Preliminaries and Problem Setup</i>	202
11.3	<i>Strong Lottery Tickets by Binary Expansion</i>	204
11.4	<i>Conclusion</i>	216

12	Efficient Memorization using Threshold Networks	217
12.1	<i>Introduction</i>	217
12.2	<i>Related works</i>	220
12.3	<i>Preliminaries</i>	222
12.4	<i>Main results</i>	224
12.5	<i>Memorization with threshold networks</i>	225
12.6	<i>Bit complexity of memorization</i>	231
12.7	<i>Lower bound construction</i>	233
12.8	<i>Conclusion</i>	235
IV	Appendices	236
A	App: Tight Convergence Bounds for Random Reshuffling	237
A.1	<i>Proof of Theorem 4.8</i>	237
A.2	<i>Proof of Theorem 4.12</i>	252
A.3	<i>Proof of Corollary 4.13</i>	269
A.4	<i>Numerical results</i>	271
B	App: Random Reshuffling in Distributed and Federated Learning	272
B.1	<i>Comparisons with assumptions and rates in existing results</i>	272
B.2	<i>More detailed illustration of synchronized shuffling</i>	278
B.3	<i>Experimental results</i>	280
B.4	<i>Proofs of upper bounds</i>	282
B.5	<i>Proof of lower bound for minibatch RR (Theorem 5.8)</i>	315
B.6	<i>Proofs of helper lemmas for Appendix B.5</i>	324
B.7	<i>Proof of lower bound for local RR: homogeneous case (Theorem 5.10)</i>	334
B.8	<i>Proofs of helper lemmas for Appendix B.7</i>	344
B.9	<i>Proof of lower bound for local RR: heterogeneous case (Proposition 5.11)</i>	353

C	App: Random Reshuffling: Is Random Optimal?	362
	<i>C.1 Discussion and Future Work</i>	362
	<i>C.2 Proof of Theorem 6.5</i>	363
	<i>C.3 Proof of Theorem 6.6</i>	372
	<i>C.4 Proof of Theorem 6.7</i>	379
	<i>C.5 Proof of Theorem 6.9</i>	380
	<i>C.6 Proof of Theorem 6.10</i>	392
	<i>C.7 Proof of Theorem 6.11</i>	407
	<i>C.8 Additional Experiments</i>	413
D	App: Looped Transformers as Programmable Computers	415
	<i>D.1 Omitted proofs</i>	415
	<i>D.2 Error Analysis</i>	429
	<i>D.3 subleq is Turing Complete</i>	432
	<i>D.4 Single Instruction Set</i>	433
	<i>D.5 Calculator</i>	442
E	App: Expressive Power of Normalization Layers	445
	<i>E.1 Linear Algebra</i>	445
	<i>E.2 Width Overparameterization</i>	450
	<i>E.3 Depth Overparameterization</i>	453
	<i>E.4 Sparse Matrix Inverse</i>	459
	<i>E.5 Experiments</i>	465
F	App: Logarithmic Bounds for Strong LTH via Subset Sum Problem	468
	<i>F.1 Proof of the upper bound</i>	468
	<i>F.2 Proof of Lower Bound</i>	475
	<i>F.3 Subset sum results</i>	477

G	App: Logarithmic Bounds for Strong LTH for Binary Networks	479
	<i>G.1 Proof of Lemma 11.7</i>	479
	<i>G.2 Observation on the power of zero</i>	481
H	App: Efficient Memorization using Threshold Networks	483
	<i>H.1 Proof of Theorem 12.8</i>	483
	<i>H.2 Proof of Theorem 12.9 and Theorem 12.16</i>	496
	<i>H.3 Proofs from Section 12.6</i>	504
	References	507

LIST OF TABLES

2.1	Tight convergence rates for Shuffling-Based SGD	9
2.2	Improved convergence rates of FLIPFLOP	15
4.1	Tight convergence rates for Shuffling-Based SGD	39
6.1	Convergence rates of RANDOM RESHUFFLE (RR), SINGLE SHUFFLE (SS) and INCRE- MENTAL GRADIENT DESCENT (IGD) on strongly convex quadratics: Plain vs. FLIPFLOP	82
8.1	Size of a Transformer needed for various operations	104
11.1	Comparing our Strong LTH results for binary networks with existing results . .	201
12.1	Upper bounds for the parameters needed for a threshold network to memorize a dataset	219

LIST OF FIGURES

2.1	Gradient Descent	6
2.2	Function construction for Shuffling-Based SGD lower bounds	18
3.1	Encoder-decoder-based transformer architecture	22
3.2	Proof Sketch for Strong LTH	32
4.1	Function construction for Shuffling-Based SGD lower bounds	52
4.2	Numerical verification of our lower bound for Shuffling-Based SGD	56
6.1	(A graphical depiction of Theorem 1's proof sketch.) Assume that the minimizer is at the origin. The proof of Theorem 6.5 first shows that there exists an initialization and a sequence of permutations, such that using those, we get to the exact minimizer. Let the sequence of iterates for this run be x_{opt}^k . Consider a parallel run, which uses the same sequence of permutations, but an arbitrary initialization point. Let this sequence be x_k . The figure shows how x_{opt}^k converges to the exact optima, and the distance between x_{opt}^k and x_k decreases exponentially, leading to an exponential convergence for x_k	88
6.2	Dependence of convergence rates on the number of epochs K for quadratic functions. The figures show the median and inter-quartile range after 10 runs of each algorithm, with random initializations and random permutation seeds (note that SS and IGD exhibit extremely small variance). We set $n = 800$, so that $n \gg K$ and hence the higher order terms of K dominate the convergence rates. Note that both the axes are in logarithmic scale.	93
8.1	Sketch of the looped transformer architecture	103
8.2	Sketch of the three transformer blocks used as building blocks to implement a small instruction-set computer	111

8.3	Sketch of the read operation	113
8.4	Sketch of the write operation	118
8.5	Building blocks used to implement the OISC instruction	122
8.6	The structure of input $\mathbf{X}$, to execute FLEQ commands.	132
9.1	Our main results for expressive power of normalization layers	160
9.2	Visualization of skip connections	171
10.1	Reconstruction of a scalar weight by pruning a single layer, logarithmic width ReLU network	189
10.2	Performance of pruning our structure shown in Figure 10.1c, with and without ReLU activation, using 5 (a_i, b_i) coefficient pairs per weight. The number of pa- rameters is a function of the sparsity of each network. Results are benchmarked on MNIST.	196
11.1	Approximating a target network with high accuracy by pruning overparameter- ized random binary network	202
11.2	Diamond-shaped binary ReLU networks that used in our binary Strong LTH construction	210
11.3	Approximating powers of 2 with binary networks	210
11.4	Illustration of our random binary network constructed using our binary gadgets	211
11.5	Approximating a 1-hidden layer network by pruning a binary network	213
12.1	Schematic of a threshold neural network that can memorize any δ -separated dataset of size n in d dimensions	226
12.2	Probability of a random hyperplane passing in between two given points	227
12.3	Visual depiction of partitioning samples in subsets	229
12.4	Visual depiction of our lower bound construction for threshold network memo- rization capacity	234

A.1	Proof overview - upper bound for RR	238
A.2	Proof overview - lower bound for RR	253
A.3	Function construction for Shuffling-Based SGD lower bounds	254
A.4	Numerical verification of RR lower bounds	271
B.1	Comparison between minibatch RR and local RR, with and without SYNC SHUF. Best viewed in color. The algorithm versions with SYNC SHUF converge faster. Also note the performance degradation as B increases, as expected by our theory.	359
B.2	Comparison of minibatch RR, local RR, and single-machine RR. Best viewed in color. The large-epoch performance of local RR becomes similar to that of single-shuffle RR as B becomes closer to N	360
B.3	Comparison of with-replacement (SGD) and without-replacement (RR) versions. Best viewed in color. Without-replacement versions converge faster than with- replacement versions, at least in our problem instance. Also note that the two versions perform similarly in the small-epoch regime, which supports our theoretical findings.	361
C.1	Dependence of convergence rates on the number of epochs K for logistic regres- sion. The figures show the median and inter-quartile range after 10 runs of each algorithm, with random initializations and random permutation seeds (note that IGD exhibits extremely small variance). We set $n = 800$, so that $n \gg K$ and hence the higher order terms of K dominate the convergence rates. Note that both the axes are in logarithmic scale.	414
D.1	The structure of input $\mathbf{X}$	434
E.1	A visualization of the skip connections. $\mathbf{x}_i$ is the i -th partition of the input $\mathbf{x}$. . .	453
E.2	Numerical verification of our constructions for BatchNorm parameters on data created by a random neural network	465

E.3	Numerical verification of our constructions for BatchNorm parameters when target network is sparse	466
F.1	Approximating a single neuron $\sigma(\mathbf{w}^T x)$: A diagram showing our construction to approximate a single neuron $\sigma(\mathbf{w}^T x)$. We construct the first hidden layer with d blocks (shown in blue), where each block contains $k = O\left(\log \frac{d}{\epsilon}\right)$ neurons. We first pre-process the weights by pruning the first layer so that it has a block structure as shown. For ease of visualization, we only show two connections per block, i.e., each neuron in the i^{th} block is connected to x_i and (before pruning) the output neuron. We then use Lemma 10.7 to show that second layer can be pruned so that i^{th} block approximates $w_i x_i$. Overall, the construction approximates $\mathbf{w}^T \mathbf{x}$. Note that, after an initial pre-processing of the first layer, we only prune the second layer so that we can re-use the weights to approximate other neurons in a layer.	469
F.2	Approximating a layer $\sigma(\mathbf{W}\mathbf{x})$: A diagram showing our construction to approximate a layer. Let $\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_d$ be the d rows of $\mathbf{W}$, i.e., the weights of d neurons. Our construction has an additional hidden layer, which contains d blocks (highlighted in blue), where each unit contains $k = O(\log(\frac{d}{\epsilon}))$ neurons. We first pre-process the weights by pruning the first layer so that it has a block structure as shown. For ease of visualization, we only show two connections per block, i.e., each neuron in the i^{th} block is connected to x_i and (before pruning) all the output neurons.	471

H.1	Left: We can compute each of $\langle \mathbf{z}^{(1)}, \mathbf{b}_k \rangle$ using subnetworks shown in blue, that we can put in parallel to output $\mathbf{z}^{(2)}$. Right: A subnetwork that implements $\langle \mathbf{z}^{(1)}, \mathbf{b}_1 \rangle$. This, in turn can be broken down as $\langle \mathbf{z}^{(1)}, \mathbf{b}_1 \rangle_{\oplus} = (z_1^{(1)} \cdot \mathbf{b}_{1,1}) \oplus (z_2^{(1)} \cdot \mathbf{b}_{1,2}) \oplus \cdots \oplus (z_{d^{(1)}}^{(1)} \cdot \mathbf{b}_{1,d^{(1)}})$. Then, the figure on the right shows how the network computes $\langle \mathbf{z}^{(1)}, \mathbf{b}_1 \rangle$ using a binary tree structure (see the orange lines in the figure). In the figure, we take $d^{(1)} = 4$. The network first computes $z_1^* = (z_1^{(1)} \cdot \mathbf{b}_{1,1}) \oplus (z_2^{(1)} \cdot \mathbf{b}_{1,2})$ and $z_2^* = (z_3^{(1)} \cdot \mathbf{b}_{1,3}) \oplus (z_4^{(1)} \cdot \mathbf{b}_{1,4})$. Then, it computes $z_1^* \oplus z_2^*$, as shown in the figure.	492
H.2	Construction for information theoretic lower bound for model size needed for memorization	504
H.3	Construction for model class that matches the information theoretic lower bound for memorization	505

LIST OF ALGORITHMS

2.1	Stochastic Gradient Descent With-Replacement (SGD)	7
2.2	SGD without-replacement	8
2.3	Shuffling-based SGD or Random Reshuffling (RR)	8
3.1	SUBLEQ(a, b, c)	26
5.1	Local RR (with and without SYNC SHUF)	65
5.2	Minibatch RR (with and without SYNC SHUF)	66
6.1	Shuffling-based SGD variants	82
8.1	Looped Transformer	108
8.2	SUBLEQ(a, b, c)	121
8.3	A sample program for executing a basic calculator functionality using our Transformer-based OISC construction	142
8.4	Pseudocode for running Newton's algorithm for Matrix inversion for T iterations.	146
8.5	Program to compute the approximate inverse using our Unified Attention Based Computer	147
8.6	Power Iteration	147
8.7	Newton's algorithm to compute inverse square root: $1/\sqrt{S}$	148
8.8	Program to simulate Power Iteration using our Unified Attention Based Computer	149
8.9	Program to simulate SGD using our Unified Attention Based Computer	151
8.10	Backpropagation	152
8.11	Program to simulate Backpropagation for two layer Neural Networks	154
8.12	Program to simulate SGD using our Unified Attention Based Computer	155
D.1	Translation for sub(a, n)	433

Part I

Introduction

1 MOTIVATION

Machine learning has revolutionized various fields including computer vision, natural language processing, and speech recognition. The success of machine learning models has been driven by two important components: SGD-based fast training algorithms and powerful neural architectures. In recent years, the field of machine learning has witnessed significant empirical advances in both these areas.

Stochastic Gradient Descent (SGD) is the most widely used optimization algorithm in deep learning due to its fast convergence rate, low generalization error, low memory requirements, and ease of parallelization and distribution. The shuffling-based variant of SGD is particularly popular because of its convergence rate which is even faster than vanilla SGD, while enjoying the same advantages as vanilla SGD mentioned above (Bottou, 2009; Recht and Ré, 2012a; Recht and Re, 2013). Shuffling-based SGD, also known as SGD without-replacement or Random Reshuffling (RR), has been known to be faster than its with-replacement counterpart for over a decade (Bottou, 2009), and yet it has eluded good theoretical understanding due to some technical reasons that are explained later in this thesis. It is of high importance to develop a tight theoretical analysis of this algorithm to not only get guarantees on its convergence rates, but also because understanding why this algorithm is better will help us design even faster algorithms.

In addition to fast training algorithms, the field of machine learning has also witnessed significant advancements in neural architectures. While traditional architectures like Convolutional neural networks (LeCun et al., 1998) for image tasks and Recurrent neural networks (Hochreiter and Schmidhuber, 1997) for language and sequence tasks are domain specific; newer architectures like Transformers (Vaswani et al., 2017) have shown exceptional performance in a diverse range of tasks including language (Brown et al., 2020), image (Dosovitskiy et al., 2021), video (Neimark et al., 2021), and sequence (de Souza Pereira Moreira et al., 2021) tasks. Special layers like BatchNorm (Ioffe and Szegedy, 2015a),

LayerNorm (Ba et al., 2016), etc. have also been proposed recently that have significantly improved the convergence rates and generalization of deep neural networks. Further, techniques like sparsity in neural networks has shown various benefits like faster training (Frankle and Carbin, 2018a), better generalization (Srivastava et al., 2014), and lower communication costs (Wang et al., 2021a). While these techniques have been widely accepted in practice, we only have a very limited theoretical understanding of these architectures and techniques; and similar to the case of Shuffling-based SGD, developing an in-depth theoretical understanding of these architectures and techniques will help provide concrete guarantees and provide new ideas on improving upon these.

While there are many interesting theoretical questions open on the two topics described above, in this thesis we scope ourselves to some key open questions and successfully provide tight answers for those.

1.1 List of Papers

This thesis presents the following papers on *Shuffling-based SGD*

1. "Closing the convergence gap of sgd without replacement", ICML 2020 (Rajput et al., 2020),
2. "Minibatch vs local sgd with shuffling: Tight convergence bounds and beyond", ICLR 2022 (Yun et al., 2022), and
3. "Permutation-based SGD: Is random optimal?", ICLR 2022 (Rajput et al., 2022);

as well as the following papers on *The expressive power of modern neural architectures*

1. "Looped transformers as programmable computers", ICLR 2023 ME-FoMo Workshop (Giannou et al.),
2. "The expressive power of tuning only the norm layers", arXiv preprint (Giannou et al., 2023),

3. "Optimal lottery tickets via subset sum: Logarithmic over-parameterization is sufficient", NeurIPS 2020 (Pensia et al., 2020),
4. "Finding nearly everything within random binary networks", AISTATS 2022 (Sreenivasan et al., 2022), and
5. "An exponential improvement on the memorization capacity of deep threshold networks", NeurIPS2021 (Rajput et al., 2021)

Apart from the topics and papers presented in this dissertation, the author has also worked on Generative Recommender Systems (Rajput et al., 2023), fine-tuning Large Language Models (Dinh et al., 2022b), unsupervised language translation using CLIP (Dinh et al., 2022a), backdoor vulnerabilities in Federated Learning (Wang et al., 2020), theoretical guarantees of data augmentation (Rajput et al., 2019a), implicit bias of adversarial training (Charles et al., 2019), and Byzantine robustness (Rajput et al., 2019b).

2 SHUFFLING-BASED STOCHASTIC GRADIENT DESCENT

In this chapter, we summarize our results on the convergence rates of Shuffling-Based SGD in centralized and distributed settings, as well as our analysis on the optimality of random permutations, where we propose a new algorithm that speeds up the convergence of Shuffling-Based SGD on quadratic loss functions.

2.1 Preliminaries

Stochastic Gradient Descent is a first-order optimization method which is considered one of the main workhorses of modern machine learning. Here, first order method means that it only uses gradient information of the function, *i.e.*, the first order derivative information. To see how SGD can be used for training, we first view the task of training a model as a finite sum minimization problem of the following form:

$$\min_{\mathbf{x}} \left(F(\mathbf{x}) := \frac{1}{n} \sum_{i=1}^n f_i(\mathbf{x}) \right). \quad (2.1)$$

Here, $f_i(\mathbf{x})$ is the loss of the model on the i -th training point, where $\mathbf{x}$ are the parameters (weights and biases) of the model. Hence, the minimization problem above is equivalent to finding the set of weights and biases that minimize the average loss of the model over the training dataset.

If the function F is differentiable, then one intuitive way of finding its (local) minima is to use Gradient Descent (GD): Start from some initialization $\mathbf{x}_0$, and then iteratively follow the path of steepest descent, which is given by the direction opposite to the gradient (see Figure 2.1). This can be written mathematically as

$$\mathbf{x}_{t+1} := \mathbf{x}_t - \alpha \nabla F(\mathbf{x}_t),$$

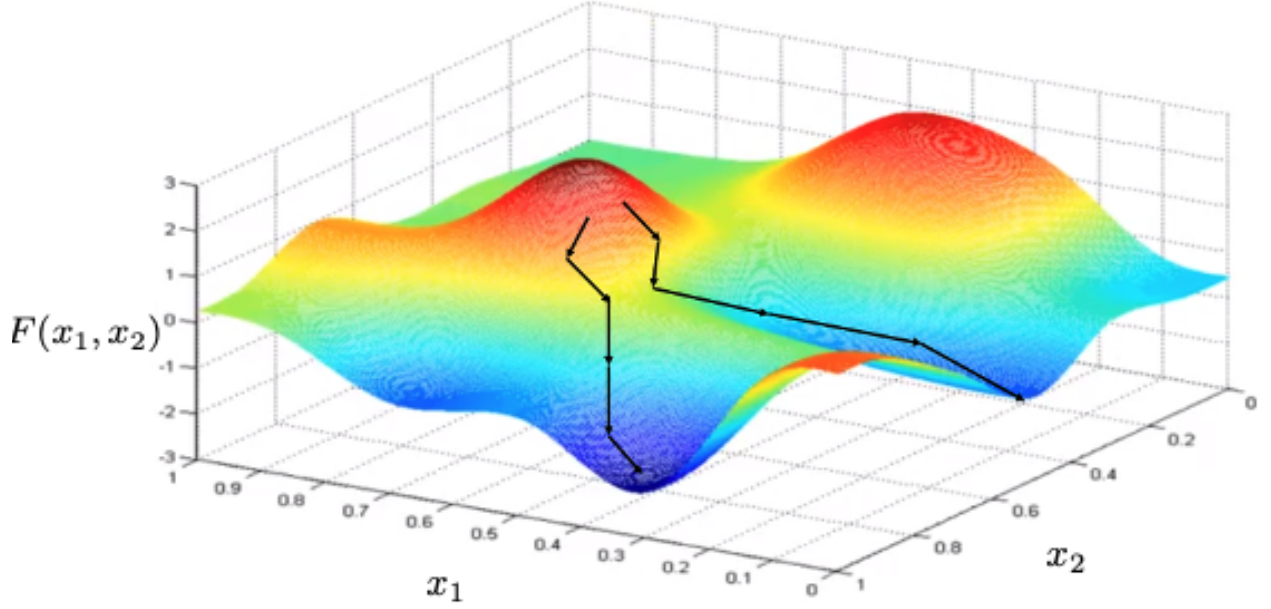


Figure 2.1: Graphical depiction of Gradient Descent from two different initializations. The arrows indicate the direction of steepest descent, which is the direction opposite to the direction of gradient. We can see that the process ends up in a local minima. Figure taken from (Ng, 2023).

for $t = 0, 1, \dots$. Here α is a tunable parameter called step-size and it can either be fixed to be a constant or can be varied during the training. While this algorithm can indeed find local and global minima under relatively weak assumptions (Nocedal and Wright, 2006), each iteration here needs the computation of the $\nabla F(\mathbf{x})$, which in-turn needs the computation of the n gradients $\nabla f_i, i = 1, 2, \dots$ (see Eq. (2.1)). This can make each iteration prohibitively expensive for modern big datasets which can have extremely large n (Deng et al., 2009; Stephen et al., 2017; Ni et al., 2019).

One way to overcome this problem is to get a *stochastic* estimate of the gradient $\nabla F(\mathbf{x})$ by computing the gradient on only one randomly sampled training point (Robbins and Monro, 1951). If the sampling is done uniformly at random from the n points in the dataset, it is easy to see that this is an *unbiased* estimate of the true gradient $\nabla F(\mathbf{x})$:

$$\mathbb{E}_{i \sim \mathcal{U}([n])}[\nabla f_i(\mathbf{x})] = \nabla F(\mathbf{x}), \quad (2.2)$$

where $\mathcal{U}([n])$ is the uniform distribution over the set $\{1, \dots, n\}$. This is the SGD algorithm,

formally described in Algorithm 2.1.

Algorithm 2.1 Stochastic Gradient Descent With-Replacement (SGD)

Input: Initialization (possibly random) $\mathbf{x}_0$, step size α , number of iterations T

```

1: for  $t = 1, \dots, T$  do
2:   Randomly sample an index  $i$  from the set  $\{1, \dots, n\}$ .
3:   Take a gradient descent step,  $\mathbf{x}_{t+1} := \mathbf{x}_t + \alpha \nabla f_i(\mathbf{x}_t)$ .
4: end for

```

There are many advantages of SGD that make it very well suited for modern machine learning tasks:

- **Ease of computation:** Since each iteration of SGD only works on one (or a few samples in the case of Minibatch SGD¹), each iteration is fast, and the computational and memory requirements are also low.
- **First order information:** The fact that SGD only uses first-order information is advantageous for two reasons. Firstly, this means that the only requirement for this method to work is that the loss function should be differentiable (or has a subgradient), which is satisfied by most deep networks and loss functions. The other benefit is that computing the gradients for networks requires time and space which is linear in the number of parameters, as opposed to say second-order methods, which can take time and space which is quadratic in the number of parameters.
- **Generalization:** SGD has been known to output models that achieve good generalization, which means that such models perform well even on the test dataset. This has been observed in practice (better generalization than even GD) (Zhang et al., 2017a; Neyshabur et al., 2014), as well as studied theoretically (Hardt et al., 2016; Jiang et al., 2022).

¹Minibatch SGD is an algorithm that interpolates between GD and SGD, where at each iteration instead of sampling a single point from the dataset, a set of points (called a *minibatch*) are sampled. This increases the per-iteration time complexity but reduces the variance in the stochastic estimation of true gradient. Hence, the size of the minibatch can be used to tradeoff between the number of iterations needed for convergence and the time needed per iteration.

- **Parallelizable:** Another advantage for the minibatch version of SGD is that it can easily be parallelized over several cores or devices (Krizhevsky et al., 2017). Apart from speeding up the training, this can also be used to build frameworks to maintain privacy of user data, for instance, in Federated Learning (McMahan et al., 2017).

Note that in Step 2, Algorithm 2.1 samples the indices from the entire set $\{1, \dots, n\}$ uniformly at random at each iteration. In other words, it uses sampling *with*-replacement at each iteration. This is the version that is very well-studied in literature and (relatively) easy to analyze theoretically since it is unbiased (see (2.2)). However, most practical implementations of SGD use sampling *without*-replacement. The resulting algorithm is as follows:

Algorithm 2.2 SGD without-replacement

Input: Initialization $\mathbf{x}_0$, step size α , number of iterations T

```

1: for  $k = 1, \dots, \lfloor T/n \rfloor$  do
2:   Set  $\mathcal{I} := \{1, \dots, n\}$ 
3:   for  $i = 1, \dots, n$  do
4:     Randomly sample an element  $i$  from  $\mathcal{I}$ . Remove  $i$  from  $\mathcal{I}$ 
5:      $\mathbf{x}_{t+1} := \mathbf{x}_t - \alpha \nabla f_i(\mathbf{x}_t)$ 
6:   end for
7: end for

```

$\left. \begin{array}{l} \text{lines 2-5} \end{array} \right\} \text{Epoch } k$

This has an equivalent implementation that is more recognizable, and that uses random permutations or *shufflings* of $\{1, \dots, n\}$:

Algorithm 2.3 Shuffling-based SGD or Random Reshuffling (RR)

Input: Initialization $\mathbf{x}_0$, step size α , number of iterations T

```

1: for  $k = 1, \dots, \lfloor T/n \rfloor$  do
2:    $\sigma =$  a random permutation of  $\{1, \dots, n\}$ 
3:   for  $i = 1, \dots, n$  do
4:      $\mathbf{x}_{t+1} := \mathbf{x}_t - \alpha \nabla f_{\sigma_i}(\mathbf{x}_t)$ 
5:   end for
6: end for

```

$\left. \begin{array}{l} \text{lines 2-5} \end{array} \right\} \text{Epoch } k$

SGD without-replacement is also known as Shuffling-based SGD or Random Reshuffling (RR).

Remark 2.1. For conciseness, in this thesis we abbreviate SGD with-replacement simply as SGD and its without-replacement counterpart, i.e., Random Reshuffling, as RR.

It has been known empirically for more than a decade that RR converges faster than SGD Bottou (2009). In this thesis, we theoretically prove why this is the case for strongly convex functions, and explore if random permutations are indeed the optimal shuffling schemes.

2.2 Summary of Results

2.2.1 Tight Convergence rates for Random Reshuffling

L is strongly convex quadratic		L is a general strongly convex smooth function	
Lower bound, (Safran and Shamir, 2020)	$\Omega\left(\frac{1}{n^2K^2} + \frac{1}{nK^3}\right)$	Lower bound, (Safran and Shamir, 2020)	$\Omega\left(\frac{1}{n^2K^2} + \frac{1}{nK^3}\right)$
Upper bound, (Haochen and Sra, 2019a)	$\tilde{\mathcal{O}}\left(\frac{1}{n^2K^2} + \frac{1}{K^3}\right)$	Upper bound, (Nagaraj et al., 2019)	$\tilde{\mathcal{O}}\left(\frac{1}{nK^2}\right)$
Our upper bound, Theorem 2.6	$\tilde{\mathcal{O}}\left(\frac{1}{n^2K^2} + \frac{1}{nK^3}\right)$	Our lower bound, Theorem 2.7	$\Omega\left(\frac{1}{nK^2}\right)$

Table 2.1: Comparison with the previous state-of-the-art results of our lower and upper bounds on the convergence error for RR. Here n is the number of component functions and K is the number of epochs. Note that the total number iterations is $T = nK$. Our matching bounds establish optimal rates for RR. For comparison, note that SGD converges at a rate of $\mathcal{O}(1/nK)$. We note that the $\tilde{\mathcal{O}}(\cdot)$ notation hides logarithmic factors.

In this thesis, we prove tight convergence rates for RR when applied on strongly convex smooth functions. In doing so, we discover an interesting phenomenon: RR converges faster on quadratics than general, strongly convex functions. This we think, is the main reason why earlier work was not able to establish matching upper and lower bounds for this algorithm, since we think they were trying to prove upper bounds for general strongly convex functions, and the constructions for lower bounds used quadratic functions. This is surprising also because SGD does not exhibit any such speedup for quadratic functions.

Assumptions Recall that the optimization problem we consider takes the following form:

$$\min_{\mathbf{x}} \left(F(\mathbf{x}) := \frac{1}{n} \sum_{i=1}^n f_i(\mathbf{x}) \right).$$

To prove convergence rates, we make the following assumptions:

Assumption 2.2. (*Convexity of Components*) f_i is convex for all $i \in [n]$.

Assumption 2.3. (*Strong Convexity*) F is strongly convex with strong convexity parameter μ , that is

$$\forall \mathbf{x}, \mathbf{y} : F(\mathbf{y}) \geq F(\mathbf{x}) + \langle \nabla F(\mathbf{x}), \mathbf{y} - \mathbf{x} \rangle + \frac{\mu}{2} \|\mathbf{y} - \mathbf{x}\|^2$$

Assumption 2.4. (*Lipschitz Gradients*) The functions f_i are L -smooth, that is

$$\forall i, \mathbf{x}, \mathbf{y} : \|\nabla f_i(\mathbf{x}) - \nabla f_i(\mathbf{y})\| \leq L \|\mathbf{x} - \mathbf{y}\|.$$

In addition, since we prove upper bound for quadratic functions, we assume the following:

Assumption 2.5. F is a quadratic function

$$F(\mathbf{x}) = \frac{1}{2} \mathbf{x}^T \mathbf{H} \mathbf{x} + \mathbf{b}^T \mathbf{x} + c,$$

where $\mathbf{H}$ is a positive semi-definite matrix.

We denote the number of component functions in (2.1) by n , the number of epochs by K , and the total number of iterations by T . Note that $T = nK$. The informal versions of our theorems are provided below, and the comparison with earlier works is done in Table 2.1.

Theorem 2.6. *Under Assumptions 2.2-2.5, let the step size of RR be $\alpha = \frac{8 \log nK}{nK\mu}$ and the number of epochs be $K \geq 128 \frac{L^2}{\mu^2} \log nK$. Then, after $T = nK$ iterations RR achieves the following rate*

$$\mathbb{E}[\|\mathbf{x}_T - \mathbf{x}^*\|^2] = \tilde{\mathcal{O}}\left(\frac{1}{n^2 K^2} + \frac{1}{nK^3}\right),$$

where $\tilde{\mathcal{O}}(\cdot)$ hides logarithmic factors.

This matches the existing lower bounds for the convergence rate of RR on quadratic functions (Safran and Shamir, 2020). Note that this is much faster than SGD that converges at the rate $\mathcal{O}(1/nK)$ (Bubeck et al., 2015).

Next we show that the existing upper bounds for the convergence rates of RR on general strongly convex functions (Nagaraj et al., 2019) are indeed tight by proving the following lower bound which uses a non-quadratic, strongly-convex function F .

Theorem 2.7. *There exists a strongly convex function F that is the sum of n smooth convex functions, such that for any step size $\frac{1}{nK} \leq \alpha \leq \frac{C}{n}$, the error after $T = nK$ total iterations of RR satisfies*

$$\mathbb{E}[\|\mathbf{x}_T - \mathbf{x}^*\|^2] = \Omega\left(\frac{1}{nK^2}\right).$$

Note that in both the theorems above, we have ignored the dependence on L and μ . Some papers after ours indeed improved upon our results in terms of dependence on these factors. We mention these later in Chapter 7.

2.2.2 Random Reshuffling in Distributed Setup (Federated Learning)

Similar to SGD, RR also enjoys the benefit that it can be easily parallelized over multiple machines. A particularly useful and interesting case of this is Federated Learning McMahan et al. (2017). In Federated Learning, each machine is assumed to have its own private dataset, and to preserve this privacy, the data is not shared with other machines. Instead, the machines compute gradients for a shared model on their private datasets and send

back the new local model after performing the updates on the global model using these gradients. Then, the central server averages all the updates and sends back the updated and synchronized global model to all the machines. This process continues in such cycles until convergence. Note that here as well, the machines can sample with or without replacement. To reduce the amount of communication done, each machine performs some $B \geq 1$ iterations of RR or SGD before communicating back to the central server and syncing the models. We state our main theorems next.

For the Federated Learning setup, we assume that there are M machines, each with n component functions (each component function being the loss on a data point). The loss function at machine m is hence,

$$F^m(\mathbf{x}) := \frac{1}{n} \sum_{i=1}^n f_i^m(\mathbf{x}),$$

and the overall loss we want to minimize is

$$F(\mathbf{x}) := \frac{1}{M} \sum_{m=1}^M F^m(\mathbf{x}).$$

We assume that the machines perform B iterations of RR before communicating their updated models back to the central server and receiving back the synced global model. We make the following assumptions about the functions:

Assumption 2.8 (Intra-machine deviation). *There exists $\nu \geq 0$ such that for all $m \in [M]$ and $i \in [N]$,*

$$\|\nabla f_i^m(\mathbf{x}) - \nabla F^m(\mathbf{x})\| \leq \nu, \text{ for all } \mathbf{x} \in \mathbb{R}^d.$$

Assumption 2.9 (Objective-wise inter-machine deviation). *There exist $\tau \geq 0$ and $\rho \geq 1$ such that*

$$\frac{1}{M} \sum_{m=1}^M \|\nabla F^m(\mathbf{x})\| \leq \tau + \rho \|\nabla F(\mathbf{x})\|, \text{ for all } \mathbf{x} \in \mathbb{R}^d.$$

Theorem 2.10. *Consider running RR in the Federated Learning setup under Assumptions 2.2,*

2.3, 2.8, and 2.9, using a step-size $\eta = \frac{\log(MnK^2)}{\mu n K}$ for epochs $K \geq 7\rho\kappa \log(MnK^2)$. Then, with probability at least $1 - \delta$,

$$F(\mathbf{x}_T) - F^* \leq \frac{F(\mathbf{x}_0) - F(\mathbf{x}^*)}{MnK^2} + \tilde{O}\left(\frac{L^2}{\mu^3} \left(\frac{\nu^2}{MnK^2} + \frac{\nu^2 B}{n^2 K^2} + \frac{\tau^2 B^2}{n^2 K^2}\right)\right). \quad (2.3)$$

Under similar assumptions, SGD version of Federated Learning gives a rate of $\tilde{O}(\frac{L\nu^2}{\mu^2 MnK} + \frac{L^2\nu^2 B}{\mu^3 n^2 K^2} + \frac{L^2\tau^2 B^2}{\mu^3 n^2 K^2})$. We see that RR outperforms SGD in the large epoch case, when $K \geq L/\mu$.

We also show that the convergence rate in the theorem above is tight by proving lower bounds which match the upper bound above. We discuss this in detail later in the thesis.

2.2.3 On the Optimality of Random Permutations in Shuffling-Based SGD

The previous two subsections talk about using RR in centralized and distributed setup. We see that shuffling-based SGD that uses random permutations is better than vanilla SGD with-replacement. This leads to the important fundamental question of whether *random* permutations are optimal, or do there exist well-crafted (and easy to find) permutations that can lead to a faster convergence rate than random permutations. In this thesis we will see that the answer is not straightforward, and that in some cases, we can easily find permutations that lead to faster convergence than random shuffling.

Our first result says that in 1-dimension, and for Hessian-smooth loss functions, there always exist sequence of permutations that lead to exponentially fast convergence.

Assumption 2.11 (Component Hessian-smoothness). *$f_i(x)$'s have L_H -smooth second derivatives, that is,*

$$\forall x, y : |\nabla^2 f_i(x) - \nabla^2 f_i(y)| \leq L_H |x - y|.$$

Theorem 2.12. *Let Assumptions 2.2, 2.3, 2.4, and 2.11 hold. If $\alpha = \frac{\mu}{8n(L^2 + L_H G)}$, then there exists a sequence of permutations $\sigma^1, \sigma^2, \dots, \sigma^K$ such that using those permutations in Algorithm 2.3*

(instead of using random permutations) from any initialization point x_0^1 gives the error

$$|x_n^K - x^*| \leq (D + 4n\alpha G)e^{-CK},$$

where $C = \frac{\mu^2}{16(L^2 + L_H G)}$. Here D and G are instance dependent constants.

This is a very positive result, which provides some hope that this might be true for higher dimensional functions as well. Unfortunately, this is not the case, because we were able to construct functions in higher dimensions for which any permutation-based algorithm can only achieve polynomial convergence rates.

Theorem 2.13. *For any $n \geq 4$ (n must be even), there exists a $2n + 1$ -dimensional strongly convex function F which is the mean of n convex quadratic functions, such that for every permutation-based algorithm with any step size,*

$$\|\mathbf{x}_T - \mathbf{x}^*\|^2 = \Omega\left(\frac{1}{n^3 K^2}\right).$$

Note that even though this shows that there can not be algorithms that achieve exponential convergence rates, there can still be a polynomial improvement in the convergence rates by generating better permutations. We confirm that that is indeed the case by proposing a new algorithm, FLIPFLOP, that improves the convergence rates of Random Reshuffling.

The way that FLIPFLOP works is rather simple: every even epoch uses the flipped (or reversed) version of the previous epoch's permutation. The intuition behind why FLIPFLOP leads to faster convergence is as follows. Towards the end of an epoch, the contribution of earlier gradients gets attenuated. To counter this, we flip the permutation for the next epoch so that every function's contribution is diluted (approximately) equally over the course of two consecutive epochs. FLIPFLOP demonstrates that finding better permutations for specific classes of functions might be computationally easy. This technique

not only improves the convergence rates for Random Reshuffling, but also two other related algorithms: Single Shuffle (where a random permutation is sampled at the first epoch, and the same permutation is reused in every epoch), and Iterative Gradient Descent (where the permutation is simply taken to be $1, 2, \dots, n$). We summarize FLIPFLOP’s convergence rates in Table 2.2 and report the results of numerical verification later in the thesis.

	Plain	with FLIPFLOP
Random Reshuffling	$\Omega\left(\frac{1}{n^2 K^2} + \frac{1}{n K^3}\right)$	$\tilde{O}\left(\frac{1}{n^2 K^2} + \frac{1}{n K^5}\right)$
Single Shuffle	$\Omega\left(\frac{1}{n K^2}\right)$	$\tilde{O}\left(\frac{1}{n^2 K^2} + \frac{1}{n K^4}\right)$
Iterative Gradient Descent	$\Omega\left(\frac{1}{K^2}\right)$	$\tilde{O}\left(\frac{1}{n^2 K^2} + \frac{1}{K^3}\right)$

Table 2.2: Convergence rates of Random Reshuffling, Single Shuffle and Iterative Gradient Descent on strongly convex quadratics: Plain vs. FLIPFLOP. When $n \gg K$, that is when the training set is much larger than the number of epochs, which arguably is the case in practice, the convergence rates of RR, SS, and IGD are $\Omega(\frac{1}{nK^3})$, $\Omega(\frac{1}{nK^2})$, and $\Omega(\frac{1}{K^2})$ respectively. On the other hand, by combining these methods with FLIPFLOP the convergence rates become faster, *i.e.*, $\tilde{O}(\frac{1}{nK^5})$, $\tilde{O}(\frac{1}{nK^4})$, and $\tilde{O}(\frac{1}{K^3})$, respectively. The lower bounds for the plain versions of RR, SS and IGD are taken from (Safran and Shamir, 2020).

2.3 Methodology

Despite the widespread popularity of RR, it has proven to be notoriously difficult to analyze theoretically Recht and Ré (2012a). The main reason for this is that unlike SGD, where the stochastic gradients at each iteration are an unbiased estimate of the true gradient, this is no longer true for RR. To see how consider the very simple example where $L(x) = \frac{1}{2}(x-1)^2 + \frac{1}{2}(x+1)^2$, that is, $n = 2$, $\ell_1(x) = (x-1)^2$, and $\ell_2(x) = (x+1)^2$. Initializing $x_0 := 0$, we see that there can be two possibilities in the first epoch, $\sigma = [1, 2]$ or $\sigma = [2, 1]$ (see Algorithm 2.3). Then, after the first iteration, $x_1 = 2\alpha$ or $x_1 = -2\alpha$ depending on the permutation. Then given x_1 , one can deduce the permutation σ , and consequently σ_2 . Thus, the gradient will be a deterministic function of x_1 . Hence, we see that in the second iteration, the gradient will not be independent of the iterate and the functions seen so far

in the epoch, and further that

$$\mathbb{E}_\sigma[\nabla \ell_{\sigma_2}(x_1)] \neq \nabla L(x_1).$$

This presents a problem because most convergence analysis of SGD crucially use the unbiasedness of the gradient, as written mathematically in (2.2).

Proving that RR converges faster than SGD despite this major issue could only be done very recently, in a line of works that required the development of novel techniques (Haochen and Sra, 2019a; Nagaraj et al., 2019; Rajput et al., 2020; Ahn et al., 2020; Mishchenko et al., 2020). We give an overview of some of the techniques below, and explain in detail later in the thesis.

2.3.0.1 Upper Bounds

The main structure of RR that we leverage for proving convergence rates is the fact that over every epoch, each data point is seen exactly once. Then, the analysis proceeds by considering the progress made by RR over an entire epoch, rather than looking at individual steps (as is the case in the analysis of SGD). In particular, we show the following: Consider an epoch that starts at $\mathbf{x}_1$, and at the i -th iteration in the epoch, the iterate is $\mathbf{x}_i$. Then, we prove that that n steps of RR in an epoch well approximate n steps of gradient descent, that is, $-\sum_{i=1}^n \alpha \nabla \ell_{\sigma_i}(\mathbf{x}_i) \approx -n\alpha \nabla \ell_{\sigma_1}(\mathbf{x}_1)$. Note that this can be made concrete if we show that $\|\mathbf{x}_1 - \mathbf{x}_i\|$ is small. This is in fact a key lemma in the proofs, where we essentially show that $\|\mathbf{x}_1 - \mathbf{x}_i\| = \tilde{\mathcal{O}}(\alpha\sqrt{i})$, that is, $\mathbf{x}_i$ does not deviate too much from $\mathbf{x}_1$ as i grows: it deviates sublinearly in i . In our first paper Rajput et al. (2020), covered in Chapter 4, we prove this using a very interesting, but complex technique that uses iterate coupling originally proposed by Nagaraj et al. (2019). However, for the results that we proved in Yun et al. (2022), which are covered in Chapter 5, we simplify the analysis by using a concentration inequality from Schneider (2016) which is specifically proved for sampling items without

replacement.

For the convergence rate given in Theorem 2.12, which we prove in our paper Rajput et al. (2022) and in this thesis in Chapter 6, we use a different strategy. We first show that there exists a point of initialization near the optimizer and a sequence of permutations such that starting from that initialization and following that particular sequence of permutations will lead us exactly to the optimizer. Then, we show that even if we choose two different initializations but follow the same permutations, the iterates will come close to each other at an exponential rate. These two facts together prove that there exists a sequence of permutations such that following that sequence of permutations leads to exponentially fast convergence, regardless of the point of initialization. For proving the convergence rates for FLIPFLOP, we consider two epochs at a time instead of one epoch, and pair the i -th step in the first epoch with the $n - i$ -th step in the second epoch (since they correspond to the same component function), and show that a lot of the noise induced during the first epoch gets cancelled out during the second epoch.

2.3.0.2 Lower Bounds

To prove lower bounds that show that the convergence rates proved by us for RR are tight (in both normal, and Federated settings), we need to construct a class of functions, such that running RR on that class of functions will lead to a convergence rate that is at least as slow as what our theorems predict. To construct such classes of functions, we first see that this class of functions cannot be quadratic functions (or functions with Lipschitz-continuous Hessians) because for such functions, we actually know of faster convergence rates than general strongly convex functions (Haochen and Sra, 2019a; Rajput et al., 2020). Thus, we construct a piece-wise quadratic function, with discontinuous Hessian. We give a brief overview of the proof sketch next.

The key idea behind our lower bound has two main steps:

1. First, we prove that that the key lemma used in the proofs of our convergence rates

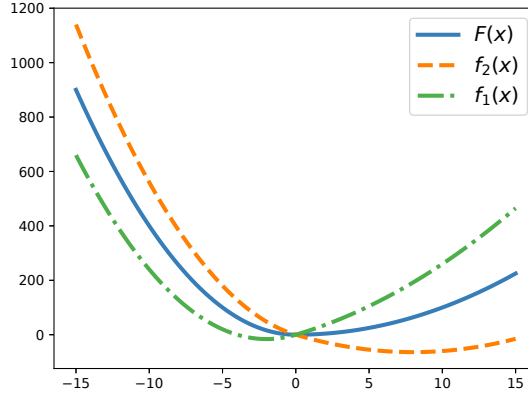


Figure 2.2: The function class used to prove the lower bounds. These functions are piecewise quadratic, and asymmetric with the functions having a larger second derivative for $x < 0$ than for $x > 0$.

is tight. Recall that this key lemma essentially states that $\|x_i - x_1\| = \tilde{O}(\alpha\sqrt{i})$. We show that this is tight, *i.e.*, for certain classes of functions we get $\|x_i - x_1\| = \tilde{\Omega}(\alpha\sqrt{i})$.

2. For this step, assume that the epoch starts close to the optimizer, that is, $x_1 \approx x^*$. Then, the deviations of x_i around x_1 mentioned in the first step are ‘symmetric’. This means that the iterates move around either side of x_1 up to a distance of approximately $O(\alpha\sqrt{n})$. Since $x_1 \approx x^*$, this means that the iterates can wander off as far as $O(\alpha\sqrt{n})$ around the optimizer in the middle part of this epoch. These iterates accumulate noisy gradients, but for quadratics most of this noise cancels out since quadratic functions are also symmetric around the minimizer. Hence, we create piecewise quadratic functions, which are not symmetric around the minimizer. The function is shown in Figure 2.2. For such functions we show that the accumulated noise does not cancel out, and we get the desired lower bounds for convergence rates.

For proving the lower bound in Theorem 2.13, we construct a class of high dimensional functions such that any permutation that leads to faster convergence along one dimension will necessarily lead to worse convergence along the other dimensions. A numerical example explaining this is as follows:

Consider a simple setting, with $n = 2$ and 2-Dimensions, x and y . Consider $F(x, y) = \frac{1}{2}f_1(x, y) + \frac{1}{2}f_2(x, y)$ such that

$$f_1(x, y) = \frac{x^2}{2} - x + y, \text{ and } f_2(x, y) = \frac{y^2}{2} - y + x.$$

Hence, $F = \frac{1}{4}(x^2 + y^2)$, and has minimizer at the origin. Each epoch has two possible permutations, $\sigma = (1, 2)$ or $\sigma = (2, 1)$. Let (x_0^k, y_0^k) denote the iterate at the beginning of the k -th epoch. Then, working out the details manually, it can be seen that regardless of the permutation, $x_0^{k+1} + y_0^{k+1} > x_0^k + y_0^k$, that is, the sum of the co-ordinates keeps increasing by a certain minimum factor. This can be used to get a bound on the error term $\|[x^K \ y^K]^\top\|^2$.

3 EXPRESSIVE POWER OF MODERN NEURAL ARCHITECTURES

In this chapter, we summarize our results on the computational power of the Transformer architecture, the approximation powers of normalization layers, the approximation power of pruning random networks, and the memorization capacity of deep threshold networks.

3.1 Preliminaries

A neuron is the basic functional unit in a neural network, and takes the form of the following function:

$$f(\mathbf{x}) := \sigma(\mathbf{w}^\top \mathbf{x} + b),$$

where $\mathbf{x} \in \mathbb{R}^d$ is the input to the neuron, $\mathbf{w} \in \mathbb{R}^d$ is the weight vector, $b \in \mathbb{R}$ is the bias, and $\sigma(\cdot)$ is the activation function. Some examples of activation functions are ReLU, $\sigma(x) = x \mathbb{1}_{x \geq 0}$; sigmoid, $\sigma(x) = \frac{e^x}{1+e^x}$; and threshold, $\sigma(x) = \mathbb{1}_{x \geq 0}$.

A feedforward neural network is an acyclic directed graph where each node is a neuron. The output of each neuron is fed as inputs to the neurons that are directly connected to it through outgoing edges in the directed graph. Generally, these neurons are stacked in layers, such that the outputs of one layer form the input to the next layer. Then, the i -th layer takes the following form

$$F_i(\mathbf{x}) = \sigma(\mathbf{W}_i^\top \mathbf{x} + \mathbf{b}_i),$$

where $\mathbf{x} \in \mathbb{R}^d$ is the input to the layer, $\mathbf{W}_i \in \mathbb{R}^{d' \times d}$ is the weight vector, $\mathbf{b} \in \mathbb{R}^{d'}$ is the bias, and $\sigma(\cdot)$ is the activation function applied coordinate-wise. Hence, an l -layer network simply takes the form

$$F(\mathbf{x}) := (F_l \circ \dots \circ F_1)(\mathbf{x}) = \sigma(\mathbf{W}_l^\top \sigma(\dots \sigma(\mathbf{W}_1^\top \mathbf{x} + \mathbf{b}_1) \dots) + \mathbf{b}_l). \quad (3.1)$$

A lot of very diverse architectures have been proposed over the past decade that roughly use the formulation in (3.1) as their backbones. This includes architectures like ResNet (He et al., 2016a), ConvNet (LeCun et al., 1998), LSTM (Hochreiter and Schmidhuber, 1997), etc. In this thesis, we consider the following architectures and network components a) Transformer architecture, b) Normalization layers, c) Sparse neural networks, and d) Threshold activated networks.

3.1.1 Transformer Architecture

The Transformer architecture was introduced by Vaswani et al. (2017) as a sequence-to-sequence model to better capture both long and short range dependencies using a single framework. It has proven to outperform all the previous sequence-to-sequence models on not just various sequence tasks like language modeling (Devlin et al., 2018), protein sequencing (Nambiar et al., 2020), etc., but recently also on image and video generation tasks (Ramesh et al., 2021; Yan et al., 2021).

This model takes as input a sequence of tokens and converts them into a sequence of embeddings using an embedding lookup table. It also concatenates positional encodings to each embedding which is used to keep track of each input embedding in the sequence. This is then fed into an attention mechanism that adds a linear combination of all the input embeddings to each input embedding, and the weight given to each input embedding is computed based on its similarity to a given input embedding, as well as their positional embeddings. A layer of this architecture is depicted graphically in Figure 3.1 and described mathematically as the function $f(\cdot)$ below:

$$\text{Attn}(\mathbf{X}) = \mathbf{X} + \sum_{i=1}^H \mathbf{V}^i \mathbf{X} \sigma_s(\mathbf{X}^\top \mathbf{K}^i \mathbf{Q}^i \mathbf{X}) \quad (3.2a)$$

$$f(\mathbf{X}) = \text{Attn}(\mathbf{X}) + \mathbf{W}_2 \sigma_r(\mathbf{W}_1 \text{Attn}(\mathbf{X}) + \mathbf{b}_1 \mathbf{1}_n^\top) + \mathbf{b}_2 \mathbf{1}_n^\top \quad (3.2b)$$

Here, $\mathbf{X} \in \mathbb{R}^{d \times n}$ is a matrix representing the n length input sequence, where each d -

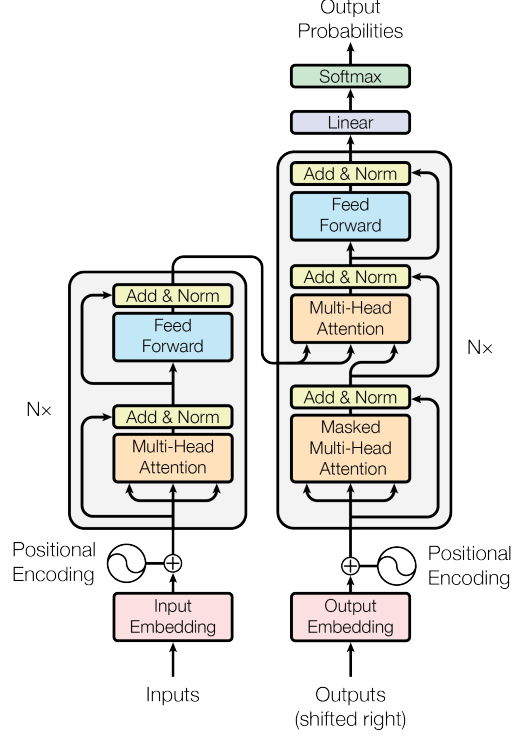


Figure 3.1: The encoder-decoder-based transformer architecture. This architecture is mathematically explained in (3.2). Figure taken from (Vaswani et al., 2017)

dimensional column contains a semantic embedding and optionally the positional embedding corresponding to its position. σ_s is the column-wise softmax operator and σ_r is the ReLU activation function. $W_1, W_2, V^i, K^i, Q^i, b_1$, and b_2 are all learnable parameters.

Recent works (Brown et al., 2020) have shown that language models based on the transformer architecture are intelligent enough to execute and follow instructions provided to them in natural language, simulate the running of python codes, and learn during inference, from data provided as input (without updating its weights). These evidence suggest that the transformer architecture can be trained to develop computational power. In this thesis, we focus on exactly this question: Can transformer based networks be coded to execute programs? What other computational tasks can they do?

3.1.2 Normalization Layers

Normalization is a technique that scales and shifts the outputs of neurons in a neural network so as to normalize them, *i.e.*, make their output distribution have zero mean and unit variance. Hence, if the output of a neuron is x , after the normalization, it becomes

$$f(x) := \gamma \frac{x - \mu}{\sigma} + \beta, \quad (3.3)$$

where the mean μ and standard deviation σ could be computed across a batch (Batch Normalization) or across the neurons in a layer (Layer Normalization, etc., and γ and β are learnable scaling and shifting parameters respectively. The first normalization technique introduced was Batch Normalization by Ioffe and Szegedy (2015a), followed by Weight Normalization (Salimans and Kingma, 2016) and Layer Normalization (Ba et al., 2016). These techniques are widely known to not just speed up training in neural networks, but also improve generalization. The expectation is that by ensuring that the layer inputs normalized, the training will be faster, similar to how input normalization typically helps training (LeCun et al., 2012; Wiesler and Ney, 2011).

These layers can also be used for finetuning pre-trained networks to fit downstream tasks (Mudrakarta et al., 2018; Huang and Belongie, 2017). The finetuning proceeds by freezing all the parameters, and only training the normalization layer parameters on each of the new tasks. This results in faster and resource efficient finetuning because a very small subset of the network parameters are being trained. A recent work by Frankle et al. (2020c) has also shown empirically that normalization layers also have significant approximation power on their own, *i.e.*, just training the normalization layers of an otherwise frozen network can lead to highly non-trivial test performance. This seems counter-intuitive, because as we see in (3.3), normalization layers can only perform scaling and shifting operations on neurons, as opposed to arbitrary linear transformations that the weight layers of neural networks perform. This leads to the question of the exact function approximation power of these

normalization layers, which we answer in this thesis.

3.1.3 Sparse Neural Networks

Sparse neural networks are neural networks where a large fraction of the weights or neurons have been removed (zeroed out). Sparsity in deep networks offers various benefits like less memory requirement and faster training (Hoeffler et al., 2021; Frankle and Carbin, 2018a), while maintaining a similar performance as that of the original, dense network.

There are two types of sparsity in neural networks: structured and unstructured sparsity. Examples of structured sparsity include pruning neurons (Vadera and Ameen, 2022), channels (in CNNs) (He et al., 2017), imposing low rank structure on weight matrices (Idelbayev and Carreira-Perpinán, 2020), or zeroing out blocks within weight matrices (Narang et al., 2017). This structure makes it easier to obtain the computational and memory efficiency gains as compared to dense networks. On the other hand, unstructured sparsity allows pruning any set of weights in a neural network. This leads to more expressive power as compared to structured sparsity, and there have been some recent hardware developments as well that take steps towards leveraging the benefits of unstructured sparsity (Pool et al., 2021).

In this thesis, we focus on unstructured sparsity. Unstructured sparsity is interesting from a theoretical perspective because unlike the usual neural network training which takes the form of continuous optimization over the weights of the network, pruning out the weights boils down to finding a set of boolean variables, which makes it a combinatorial optimization problem. This set contains a boolean variable per each weight in the network, where each of these boolean variables signify whether the corresponding weight is kept or pruned from the network. This set is also called a *mask*, since it controls which weights are masked out from the network. Note that combinatorial optimization problems are often more difficult than continuous optimization problems. In this thesis, we analyze the function approximation power of randomly initialized sparse networks: networks whose

weights have been randomly initialized and frozen, and the only operation allowed is sparsifying the network.

3.1.4 Threshold Activated Networks

Threshold activated networks are neural networks which use the threshold function, $\sigma(x) = \mathbb{1}_{x \geq 0}$ as the activation function in (3.1). Threshold networks were one of the earliest kinds of networks to be studied, with a rich line of work (McCulloh and Pitts, 1943; Borsellino and Gamba, 1961; Baum, 1988), including the famous Perceptron algorithm (Rosenblatt, 1957). Despite being studied extensively, their use in modern machine learning is very limited. This is mainly because the threshold activation is not only non-differentiable, it is also discontinuous. This makes training deep threshold networks using gradient based methods very challenging, although some recent papers propose new training algorithms for such networks (Chen et al., 2022; Ergen et al., 2023). Another fundamental challenge with these networks is that passing ‘amplitude’ information between layers is challenging. For example, in both the cases when a neuron before applying the activation is trying to output $x = 100.0$ or when the neuron before applying the activation is trying to output $x = 0.01$, the threshold activation will convert both of them to 1, hence losing the amplitude information. In particular, this makes it unclear if going deeper for threshold networks is beneficial. This is one of the questions we answer in this thesis: Does going deeper improve the memorization capacity of threshold activated networks?

3.2 Results and Proof Techniques

3.2.1 Computational Power of the Transformer Architecture

In Chapter 8, we show that transformer architecture is capable of simulating arbitrary computational commands. In particular, we prove that transformer networks can be used as universal computers by programming them with specific weights and placing

them in a loop. We demonstrate that a constant number of encoder layers can emulate basic computing blocks, such as lexicographic operations like reading and writing, non-linear functions, function calls, program counters, and conditional branches. Using these, we show how to simulate a One-Instruction Set Computer (OISC), performing matrix operations like matrix inversion and power iteration, as well as simulating backpropagation (SGD) on another network.

Theorem 3.1 (Informal). *There exists a looped transformer with less than 13 layers that can emulate a general purpose computer, a basic calculator, numerical linear algebra methods, such as approximate matrix inverse and power iteration, and in-context learning algorithms, such as SGD, on neural networks.*

These results offer theoretical support to the empirically observed phenomena like GPT-3 (Brown et al., 2020) being able to run computer programs and in-context learning.

3.2.1.1 Proof Overview

We consider the input itself as the memory of our computer. This takes the form of an $d \times n$ matrix, where d is the latent dimensionality of the transformer and n is the length of the input. This can be viewed as a sequence of n memory locations, each with dimension d . Our goal for this proof sketch will be to prove that a transformer can simulate the SUBLEQ command, as shown below. That will be sufficient to prove that transformers can simulate a universal computer because it is known that any machine that can execute a SUBLEQ command is a universal computer (Mavaddat and Parhami, 1988).

Algorithm 3.1 SUBLEQ(a, b, c)

```

1: mem[b] = mem[b] - mem[a]
2: if mem[b] ≤ 0 then
3:   goto instruction c
4: else goto next instruction
5: end if

```

In this algorithm, $\text{mem}[x]$ refers to the data or command stored at memory location x (or equivalently, the x -th column). Thus, each location can store a SUBLEQ command (in the form of a tuple (a, b, c) as shown in the algorithm above) or data. As we will see, the x -th column will not only contain data or a command, but also some other metadata that is mainly used to uniquely identify each column.

The first thing that we need is to index each column in the memory. For this we use specially constructed positional embeddings that are appended to each of the columns. These are used to uniquely identify different columns, and they can be manipulated by a feed forward layer to point to the next column. This makes it possible to implement a *program counter*. The other important job that these positional embeddings do is to point to locations from where we need to read or where we need to write data to.

We show that the attention mechanism is capable of reading and writing data that is pointed to by the positional embeddings. This is possible because the attention mechanism can be used to create linear combinations of different columns of the input, where the coefficient can depend on the positional embeddings and the contents of the columns.

Next, we show that the feedforward layers are capable of performing basic arithmetic and logical operations, that can be used to implement lines 1 and 2 of Algorithm 3.1.

Combining all these basic operations, we show that a few Transformer layers can execute a SUBLEQ command, and hence putting them in a loop like a recurrent network results in a universal computer.

3.2.2 Function Approximation Power of Normalization Layers

In Chapter 9, we explore the function powers of normalization layers like BatchNorm and LayerNorm. We prove that any given neural network can be perfectly reconstructed by only tuning the normalization layers of a wider, or deeper random network that contains only a factor of $\tilde{O}(\sqrt{d})$ more parameters (including both trainable and random). Below is an informal statement of our main theorem:

Theorem (Informal). *Let g be any fully connected, ReLU-activated neural network with l layers and width d . Then, any randomly initialized fully connected network f with l' layers and width d' , with Normalization Layers can exactly recover the network g functionally, by tuning only the Normalization Layer parameters, as long as $d'l' \geq 2d^2l$, $d' \geq d$ and $l' \geq 2l$. Further, if f has sparse weight matrices, then the total number of parameters (trainable and random) only needs to be a factor of $\tilde{O}(\sqrt{d})$ larger than the target network.*

This is quite surprising, because as opposed to weight layers of neural networks, which are able to perform arbitrary linear transformations on the layer's inputs, normalization layers can only perform scaling and shifting. We explain this in more details, along with numerical verification in Chapter 9.

3.2.2.1 Proof Sketch

We show a sketch of the proof for the case when $d' = d^2$ and $l' = 2l$. We prove our result by building a layer-by-layer reconstruction of g : For the i -th layer of g , we construct the $2i - 1$ and $2i$ layers of f such that the two layers of f together are functionally equivalent to the layer of g . In this sketch, we show a construction where the first layer of g and the first two layers of f are functionally equivalent. Without loss of generality, the same proof can be applied to all subsequent layers of g and f .

We start by using the formulation in (3.3) to write the mathematical equation for a normalization layer:

$$h_{\Gamma, \beta}(\mathbf{x}) = \Gamma \Sigma^{-1}(\mathbf{x} - \boldsymbol{\mu}) + \beta,$$

where Γ and Σ are diagonal matrices containing the corresponding γ and σ parameter for each neuron in the layer. Note that during inference, the parameters $\boldsymbol{\mu}$ and Σ are frozen.

Hence by absorbing them into Γ and β respectively, we reformulate the equation above as

$$h_{\Gamma, \beta}(\mathbf{x}) = \Gamma \mathbf{x} + \beta.$$

The first two layers of f can then be written as:

$$h_{\Gamma_2, \beta_2}(\mathbf{W}_2 \sigma(h_{\Gamma_1, \beta_1}(\mathbf{W}_1 \mathbf{x}))) = \Gamma_2 \mathbf{W}_2 \sigma(\Gamma_1 \mathbf{W}_1 \mathbf{x} + \beta_1) + \beta_2$$

where σ is the ReLU activation, and $\mathbf{W}_1, \mathbf{W}_2$ are the (randomly initialized) weight matrices of f (we set the biases to be 0 for this proof sketch). Let $\mathbf{W}_1^*$ be the weight matrix of the first layer of g . Thus to prove $f = g$, it is sufficient to show that $\Gamma_2 \mathbf{W}_2 \sigma(\Gamma_1 \mathbf{W}_1 \mathbf{x} + \beta_1) + \beta_2 = \mathbf{W}_1^* \mathbf{x}$ for all $\mathbf{x}$ with $\|\mathbf{x}\| \leq 1$.

Step 1. We set the parameters of β_1 large enough so that all ReLUs of the first layer of f are activated. For this, we only need that the parameters of the first normalization layer satisfy $\beta_1^i = \Omega(|\gamma_1^i|)$ for each neuron i . Next, we simply set $\Gamma_2 := \mathbf{I}_d$ and $\beta_2 := -\mathbf{W}_2 \beta_1$. Substituting these we get that $\Gamma_2 \mathbf{W}_2 \sigma(\Gamma_1 \mathbf{W}_1 \mathbf{x} + \beta_1) + \beta_2 = \mathbf{W}_2 \Gamma_1 \mathbf{W}_1 \mathbf{x}$. Thus, now we need to show that the system $\mathbf{W}_2 \Gamma_1 \mathbf{W}_1 \mathbf{x} = \mathbf{W}_1^* \mathbf{x}$ has a solution with respect to Γ_1 . As we will see, this solution will, in fact, be unique.

Step 2. We use the following lemma to construct a Γ_1 that satisfies the equality above.

Lemma 3.2. *Let $\mathbf{C} \in \mathbb{R}^{n \times m}$, $\mathbf{X} \in \mathbb{R}^{nm \times nm}$ be a diagonal matrix, $\mathbf{B} \in \mathbb{R}^{nm \times m}$ and $\mathbf{W}^* \in \mathbb{R}^{n \times m}$. Then, the following holds*

$$\mathbf{C} \mathbf{X} \mathbf{B} = \mathbf{W}^* \iff (\mathbf{C} \odot \mathbf{B}^T) \text{vecd}(\mathbf{X}) = \text{vec}(\mathbf{W}^*),$$

where vecd denotes the vectorization of the diagonal of a matrix, and vec denotes the row major vectorization of a matrix.

The lemma above implies that if $\mathbf{W}_2 \odot \mathbf{W}_1^T$ is invertible, then we can find unique Γ_1 ¹ that satisfies $\mathbf{W}_2 \Gamma_1 \mathbf{W}_1 = \mathbf{W}_1^*$. The following lemma says that with probability 1, this is true.

Lemma 3.3. *Let $\mathbf{A} \in \mathbb{R}^{n \times nm}$, $\mathbf{B} \in \mathbb{R}^{m \times nm}$ be two random matrices, whose elements are drawn independently from a continuous distribution; then, their Khatri-Rao product, $\mathbf{A} \odot \mathbf{B}$, is full rank with probability 1.*

For some intuition regarding the proof of this lemma, consider $\mathbf{A}$ to be an $nm \times nm$ matrix, whose elements are drawn from any continuous distribution independently. It is easy to see that this matrix is full rank, since the columns of this matrix are independent vectors, and the entries of vectors themselves come from a product distribution over $\mathbb{R}^{nm}$. For the case of the Khatri-Rao product $\mathbf{A} \odot \mathbf{B}$, where $\mathbf{A} \in \mathbb{R}^{n \times nm}$ and $\mathbf{B} \in \mathbb{R}^{m \times nm}$ however, we only have $(n + m)nm$ ‘free’ random variables, instead of $(nm)^2$. However, writing down the expression for the determinant as a polynomial in the random variables, we see that certain terms have independent coefficients from others. This helps us prove that the probability of the determinant being zero is zero, and hence the matrix is full rank.

Step 3. Repeating the same proof above for all the layers of g and the corresponding layers of f proves a layer-wise equivalence between f and g . Since f and g are just the composition of these layers, proving the equivalence layer-wise proves that $f \equiv g$.

3.2.3 Tight Bounds for the Strong Lottery Ticket Hypothesis

While most methods to sparsify a network work during or post-training, a recent groundbreaking paper by Frankle and Carbin (2018a) showed experimentally that within a randomly initialized network, there exists a subnetwork (at initialization) that when trained, with the same sparsity pattern, achieves the same accuracy as that which would have

¹Since the matrix $\mathbf{W}_2 \odot \mathbf{W}_1$ is full rank, the vector $\text{vec}(\mathbf{W}^*)$ will be a linear combination of its columns; from the Rouché-Kronecker-Capelli Theorem the solution will be unique.

been achieved by training the full network. This phenomenon, called the Lottery Ticket Hypothesis (LTH), has many advantages like faster training and smaller trained network. An even more interesting hypothesis, dubbed the Strong Lottery Ticket Hypothesis (SLTH) says that there exists a subnetwork within a randomly initialized network that achieves full accuracy even without being trained. This hypothesis, if true would imply that the task of training a network can alternatively be done by finding the correct sparsity pattern / mask for a randomly initialized network.

SLTH was first proved by Malach et al. (2020a), where the authors showed that any target network can be approximated by a pruning a polynomially larger random neural network. This result, while impressive, limits the practical viability of SLTH because the polynomially larger network could be too large. So an important question was whether this result could be improved. In our work, Pensia et al. (2020), we prove that in fact, one only needs logarithmically larger random neural network to be pruned to approximate any target network. Further, we extend this result to binary neural networks as well in Sreenivasan et al. (2022). These papers are presented in this thesis in Chapters 10 and 11 respectively. The theorem statement is as follows.

Theorem 3.4 (informal). *A randomly initialized network with width $\mathcal{O}(d \log \frac{dl}{\min\{\epsilon, \delta\}})$ and depth $2l$, with probability at least $1 - \delta$, can be pruned to approximate any neural network with width d and depth l , up to error ϵ .*

In the theorem above, we assume the initialization of each weight is done from a uniform distribution over the range $[-1, 1]$. We also prove a similar result, where each weight is initialized from a binary distribution $\{-1, 1\}$, which we provide in Chapter 11. These results together show that training a neural network can alternatively be done by only pruning a randomly initialized, logarithmically larger network and that further these trained networks would be sparse, which can have multiple benefits as mentioned earlier.

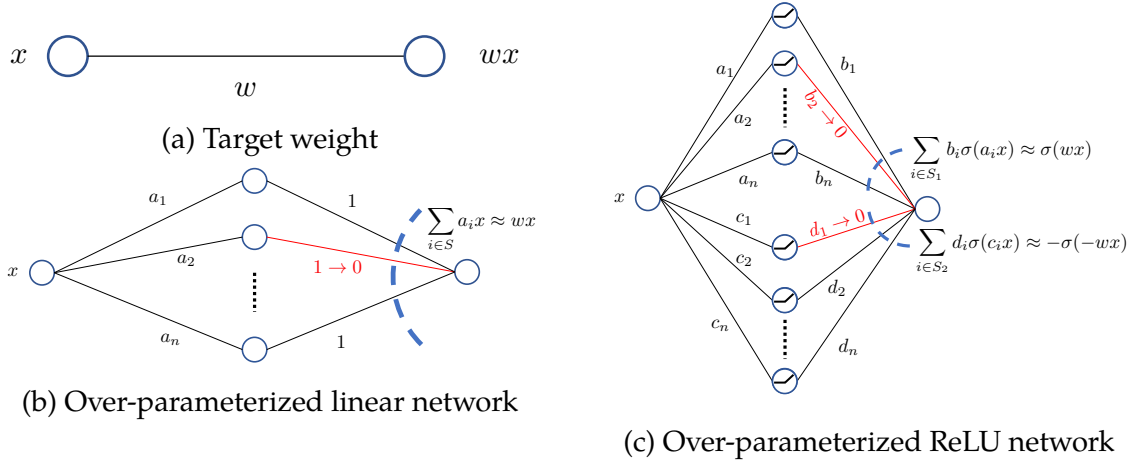


Figure 3.2: (a) Target network is a linear univariate wx . (b) The linear network shown represents the function $\sum_{i=1}^n a_i x$, where a_i are sampled uniformly from the uniform distribution over $[-1, 1]$. Pruning the weights (shown in red) results in the function $\sum_{i \in S} a_i x$, where S is set of unpruned weights. Using the result of Lueker (1998a), we get that we only need $n = \mathcal{O}(\log(1/\epsilon))$ to get an ϵ approximation of the target function wx . (c) Lemma 3.5 shows that the technique in (b) for linear networks can be extended to ReLU neural networks as well.

Proof Sketch The key idea in the theorem proof is to approximate any weight w in the target neural network by pruning a randomly initialized, 1-hidden layer ReLU network g with $\mathcal{O}(\log(1/\epsilon))$ neurons. We show this in the following lemma (also see Figure 3.2).

Lemma 3.5. (*Approximating a weight*) Let $g : \mathbb{R} \rightarrow \mathbb{R}$ be a randomly initialized network of the form $g(x) = \mathbf{b}^T \sigma(\mathbf{a}x)$, where $\mathbf{a}, \mathbf{b} \in \mathbb{R}^{2n}$, $n \geq C \log \frac{2}{\epsilon}$, and all a_i, b_i 's are drawn i.i.d. from $U[-1, 1]$. Then with probability at least $1 - \epsilon$,

$$\forall w \in [-1, 1], \quad \exists \mathbf{s} \in \{0, 1\}^{2n} : \sup_{x: |x| \leq 1} |wx - (\mathbf{b} \odot \mathbf{s})^T \sigma(\mathbf{a}x)| < \epsilon.$$

The lemma above states that we can approximate a single link with a $\mathcal{O}(\log(1/\epsilon))$ wide network. Then we show that we can approximate a neuron with d weights using $\mathcal{O}(d \log(d/\epsilon))$ wide network. Moreover, reusing weights allows us to approximate a whole layer of d neurons with a $\mathcal{O}(d \log(d/\epsilon))$ wide network. Finally, we show that if we can approximate each layer in the target network individually, we also get a good approximation

as a whole.

We also prove that the logarithmic dependence is necessary for shallow networks which have been initialized using a uniform distribution over the range $[-1, 1]$.

Theorem 3.6 (informal). *There exists a 2-layer neural network with width d which cannot be approximated to error within ϵ by pruning a randomly initialized 2-layer network, unless the random network has width at least $\Omega(d \log(1/\epsilon))$.*

Let $\mathcal{G}$ be the set of all the neural networks that can be created by pruning a randomly initialized network. Our proof strategy is a counting argument, which shows that $|\mathcal{G}|$ has to be large deterministically for it to approximate any arbitrary target network. Particularly, we show that there exists a matrix $\mathbf{W}$ such that a target network of the form $\sigma(\mathbf{W}\mathbf{x}) - \sigma(-\mathbf{W}\mathbf{x}) = \mathbf{W}\mathbf{x}$ is far from all the pruned networks in $\mathcal{G}$ with high probability, unless $|\mathcal{G}|$ is large enough. Together with the fact that $|\mathcal{G}|$ scales with the number of parameters, we get the desired lower bound.

3.2.4 Memorization Capacity of Threshold networks

An interesting question regarding the memorization capacity of threshold networks that had been open for almost three decades was the following: *What is the smallest sized threshold network that can classify perfectly every point in a given dataset with arbitrary labels?* In particular, we are interested in the minimum size (in terms of the number of weights and neurons) that a network needs to be so that it can fit a dataset of n points with arbitrary labels, where there is a distance (or margin) of at least δ between every pair of points, and the points all lie within a ball of unit norm.

While Baum (1988) was able to construct a network which had the optimal dependence on the total number of weights back in 1988, the question for the optimal dependence on the number of neurons remained open until Vershynin (2020) found the optimal dependence on neurons. The key idea was *constructing a deep threshold network that was able to memorize the*

dataset efficiently, which was novel because prior constructions all used shallow threshold networks for the reasons explained in Subsection 3.1.4. However, the construction in (Vershynin, 2020) had an exponential dependence on the minimum distance, δ . In our paper (Rajput et al., 2021), we were able to get optimal dependence on all the three parameters: number of parameters, neurons, and margin:

Theorem 3.7 (Informal). *There exists a threshold neural network with $\tilde{\mathcal{O}}\left(\frac{1}{\delta} + \sqrt{n}\right)$ neurons and $\tilde{\mathcal{O}}\left(\frac{d}{\delta} + n\right)$ weights that can memorize any δ -separated dataset of size n in d dimensions.*

Proof Sketch : To prove this theorem, we construct a network that is able to memorize the dataset. It does so in the following three steps:

1. We view the output of the first layer of the threshold network as a binary vector. We prove that if the first layer consists of neurons with random Gaussian weights, then $\mathcal{O}(\log(n)/\delta)$ neurons will be sufficient to create unique vectors for each of the inputs. Hence, after the first layer, each input has a corresponding $\mathcal{O}(\log(n)/\delta)$ dimensional, unique binary vector representing it.
2. We are then able to further compress the $\mathcal{O}(\log(n)/\delta)$ dimensional vectors to $\mathcal{O}(\log(n))$ dimensional vectors using a few sparse layers of threshold network. This requires the simulation of linear codes with threshold neurons. This is done to get a linear dependence on $1/\delta$ in the upper bounds instead of a quadratic dependence.
3. Finally, we partition the n samples into $\tilde{\mathcal{O}}(\sqrt{n})$ subsets such that each subset can be memorized by one neuron. This leads to the construction only needing $\tilde{\mathcal{O}}(\sqrt{n})$ neurons.

While the best possible dependence on the number of neurons and weights had been established in (Baum, 1988), the lower bound for dependence on the margin, δ , was unknown. We prove the following lower bound, which proves that our upper bound mentioned in the theorem above is tight, up to polynomial factors.

Theorem 3.8 (Informal). *There exists a δ -separated dataset of size $n \in \left[\frac{d^2}{\delta}, \left(\frac{1}{\delta} \right)^{\frac{d}{2}} \right]$ such that any threshold network that can memorize it needs $\tilde{\Omega} \left(\frac{1}{\sqrt{\delta}} \right)$ neurons in the first layer.*

To prove this lower bound, we reduce the problem to the following purely geometric problem: *Given n points on a unit sphere such that each pair is separated by a distance of at least δ , what is the minimum number of hyperplanes needed to ensure that each pair of points has at least one hyperplane passing between them?* To answer this question, we use tools from high-dimensional geometry, which are discussed in detail in Chapter 12.

Part II

Shuffling-Based SGD (SGD Without Replacement)

4 TIGHT CONVERGENCE BOUNDS FOR RANDOM RESHUFFLING

Random Reshuffling (RR) is a widely used variant of SGD for training models. The difference between RR and vanilla SGD is that in RR, the sampling of points is done without replacement. However, the vast majority of SGD analyses assumes data is sampled with replacement, and when the function minimized is strongly convex, an $\mathcal{O}\left(\frac{1}{T}\right)$ rate can be established when SGD is run for T iterations. A recent line of breakthrough works on RR established an $\mathcal{O}\left(\frac{n}{T^2}\right)$ convergence rate when the function minimized is strongly convex and is a sum of n smooth functions, and an $\mathcal{O}\left(\frac{1}{T^2} + \frac{n^3}{T^3}\right)$ rate for sums of quadratics. On the other hand, the tightest known lower bound postulates an $\Omega\left(\frac{1}{T^2} + \frac{n^2}{T^3}\right)$ rate, leaving open the possibility of better RR convergence rates in the general case. In this chapter, we close this gap and show that SGD without replacement achieves a rate of $\mathcal{O}\left(\frac{1}{T^2} + \frac{n^2}{T^3}\right)$ when the sum of the functions is a quadratic, and offer a new lower bound of $\Omega\left(\frac{n}{T^2}\right)$ for strongly convex functions that are sums of smooth functions.

4.1 Introduction

Stochastic gradient descent (SGD) is a widely used first order optimization technique used to approximately minimize a sum of functions

$$F(x) = \frac{1}{n} \sum_{i=1}^n f_i(x).$$

In its most general form, SGD produces a series of iterates

$$x_{i+1} = x_i - \alpha \cdot g(x, \xi_i)$$

This chapter presents our paper "Closing the Convergence Gap of SGD Without Replacement" (Rajput et al., 2020).

where x_i is the i -th iterate, $g(x, \xi_i)$ is a stochastic gradient defined below, ξ_i is a random variable that determines the choice of a single or a subset of sampled functions f_i , and α represents the step size. With- and without replacement sampling of the individual component functions are regarded as some of the most popular variants of SGD. During SGD with replacement sampling, the stochastic gradient is equal to $g(x, \xi_i) = \nabla f_{\xi_i}(x)$ and ξ_i is a uniform number in $\{1, \dots, n\}$, *i.e.*, a with replacement sample from the set of gradients $\nabla f_1, \dots, \nabla f_n$. In the case of without replacement sampling, the stochastic gradient is equal to $g(x, \xi_i) = \nabla f_{\xi_i}(x)$ and ξ_i is the i -th ordered element in a random permutation of the numbers in $\{1, \dots, n\}$, *i.e.*, a without-replacement sample.

In practice, SGD without replacement (also known as Random Reshuffling) is much more widely used compared to its with replacement counterpart, as it can empirically converge significantly faster (Bottou, 2009; Recht and Re, 2013; Recht and Ré, 2012a). However, in the land of theoretical guarantees, with replacement SGD has been the focal point of convergence analyses. This is because analyzing stochastic gradients sampled with replacement are significantly more tractable. The reason is simple: in expectation, the stochastic gradient is equal to the “true” gradient of F , *i.e.*, $\mathbb{E}_{\xi_i} \nabla f_{\xi_i}(x) = \nabla F(x)$. This makes SGD amenable to analyses very similar to that of vanilla gradient descent (GD), which has been extensively studied under a large variety of function classes and geometric assumptions, *e.g.*, see Bubeck et al. (2015).

Unfortunately, the same cannot be said for Random Reshuffling, which has long resisted non-vacuous convergence guarantees. For example, although we have long known that SGD with replacement can achieve a $\mathcal{O}\left(\frac{1}{T}\right)$ rate for strongly convex functions F , for many years the best known bounds for Random Reshuffling did not even match that rate, in contrast to empirical evidence. However, a recent series of breakthrough results on Random Reshuffling has established similar or better convergence rates than SGD with replacement.

Gürbüzbalaban et al. (2015) established for the first time that for sums of quadratics or smooth functions, there exist parameter regimes under which RR achieves an $\mathcal{O}(n^2/T^2)$

F is strongly convex quadratic		F is a general strongly convex smooth function	
Lower bound, Safran and Shamir (2020)	$\Omega\left(\frac{1}{T^2} + \frac{n^2}{T^3}\right)$	Lower bound, Safran and Shamir (2020)	$\Omega\left(\frac{1}{T^2} + \frac{n^2}{T^3}\right)$
Upper bound, Haochen and Sra (2019a)	$\tilde{\mathcal{O}}\left(\frac{1}{T^2} + \frac{n^3}{T^3}\right)$	Upper bound, Nagaraj et al. (2019)	$\tilde{\mathcal{O}}\left(\frac{n}{T^2}\right)$
Our upper bound, Theorem 4.8	$\tilde{\mathcal{O}}\left(\frac{1}{T^2} + \frac{n^2}{T^3}\right)$	Our lower bound, Theorem 4.12	$\Omega\left(\frac{n}{T^2}\right)$

Table 4.1: Comparison of our lower and upper bounds to current state-of-the-art results. Our matching bounds establish information theoretically optimal rates for SGD. We note that the $\tilde{\mathcal{O}}(\cdot)$ notation hides logarithmic factors.

rate compared to the $\mathcal{O}(1/T)$ rate of SGD with replacement sampling. In this case, if n is considered a constant, then RR becomes T times faster than SGD with replacement. Shamir (2016a) showed that for one epoch, *i.e.*, one pass over the n functions, RR achieves a convergence rate of $\mathcal{O}(1/T)$. More recently, Haochen and Sra (2019a) showed that for functions that are sums of quadratics, or smooth functions under a Hessian smoothness assumption, one could obtain an even faster rate of $\mathcal{O}\left(\frac{1}{T^2} + \frac{n^3}{T^3}\right)$. Nagaraj et al. (2019) show that for Lipschitz convex functions, RR is at least as fast as SGD with replacement, and for functions that are strongly convex and sum of n smooth components one can achieve a rate of $\mathcal{O}\left(\frac{n}{T^2}\right)$. This latter result was the first convergence rate that provably establishes the superiority of Random Reshuffling even for the regime that n is not a constant, as long as the number of iterations T grows faster than the number n of function components.

This new wave of upper bounds has also been followed by new lower bounds. Safran and Shamir (2020) establish that there exist sums of quadratics on which RR cannot converge faster than $\Omega\left(\frac{1}{T^2} + \frac{n^2}{T^3}\right)$. This lower bound gave rise to a gap between achievable rates and information theoretic impossibility. On one hand, RR on n quadratics has a rate of at least $\Omega\left(\frac{1}{T^2} + \frac{n^2}{T^3}\right)$ and at most $\mathcal{O}\left(\frac{1}{T^2} + \frac{n^3}{T^3}\right)$. On the other hand, for the more general class of strongly convex functions that are sums of smooth functions the best rate is $\mathcal{O}\left(\frac{n}{T^2}\right)$. This leaves open the question of whether the upper or lower bounds are loose. This is precisely the gap we close in this work.

Our Contributions: In this work, we establish tight bounds for RR. We close the gap between lower and upper bounds on two of the function classes that prior works have focused on: strongly convex functions that are *i)* sums of quadratics and *ii)* sums of smooth functions. Specifically, for *i)*, we offer tighter convergence rates, *i.e.*, an upper bound that matches the lower bound given by Safran and Shamir (2020); as a matter of fact our convergence rates apply to general quadratic functions that are strongly convex, which is a little more general of a function class. For *ii)*, we provide a new lower bound that matches the upper bound by Nagaraj et al. (2019). A detailed comparison of current and proposed bounds can be found in Table 4.1.

A few words on the techniques used are in order. For our convergence rate on quadratic functions, we heavily rely on and combine the approaches used by Nagaraj et al. (2019) and Haochen and Sra (2019a). The convergence rate analyses proposed by Haochen and Sra (2019a) can be tightened by a more careful analysis that employs iterate coupling similar to the one used by Nagaraj et al. (2019), combined with new bounds on the deviation of the stochastic, without-replacement gradient from the true gradient of F .

For our lower bound, we use a similar construction to the one used by Safran and Shamir (2020), with the difference that each of the individual function components is not a quadratic function, but rather a piece-wise quadratic. This particular function has the property we need: it is smooth, but not quadratic. By appropriately scaling the sharpness of the individual quadratics we construct a function that behaves in a way that Random Reshuffling cannot converge faster than a rate of n/T^2 , no matter what step size one chooses.

We note that although our methods have an optimal dependence on n and T , we believe that the dependence on function parameters, *e.g.*, strong convexity, Lipschitz, and smoothness, can potentially be improved.

4.2 Related Work

The recent flurry of work on without replacement sampling in stochastic optimization extends to several variants of stochastic algorithms beyond SGD. In (Lee and Wright, 2019; Wright and Lee, 2017), the authors provide convergence rates for random cyclic coordinate descent, establishing for the first time that it can provably converge faster than stochastic coordinate descent with replacement sampling. This work is complemented by a lower bound on the gap between the random and non-random permutation variant of coordinate descent (Sun and Ye, 2019). Several other works have focused on the random permutation variant of coordinate descent, *e.g.*, see (Gurbuzbalaban et al., 2019b; Sun et al., 2019). In (Gurbuzbalaban et al., 2019a), novel bounds are given for incremental Newton based methods. Meng et al. (2019) present convergence bounds for with replacement sampling and distributed SGD. Finally, Ying et al. (2018) present asymptotic bounds for RR for strongly convex functions, and show that with a constant step size it approaches the global optimizer to within smaller error radius compared to SGD with replacement. In (Shamir, 2016a), linear convergence is established for a without replacement variant of SVRG.

4.3 Preliminaries and Notation

We focus on using RR to approximately find x^* , the global minimizer of the following unconstrained minimization problem

$$\min_{x \in \mathbb{R}^d} \left(F(x) := \frac{1}{n} \sum_{i=1}^n f_i(x) \right).$$

In our convergence bounds, we denote by T the total number of iterations of RR, and by K the number of epochs, *i.e.*, passes over the data. Hence,

$$T = nK.$$

In our derivations, we denote by x_i^j the i -th iterate of the j -th epoch. Consequentially, we have that $x_0^{j+1} \equiv x_n^j$.

Our results in the following sections rely on the following assumptions.

Assumption 4.1. (*Convexity of Components*) f_i is convex for all $i \in [n]$.

Assumption 4.2. (*Strong Convexity*) F is strongly convex with strong convexity parameter μ , that is

$$\forall x, y : F(y) \geq F(x) + \langle \nabla F(x), y - x \rangle + \frac{\mu}{2} \|y - x\|^2$$

Assumption 4.3. (*Bounded Domain*)

$$\forall x : \|x - x^*\| \leq D.$$

Assumption 4.4. (*Bounded Gradients*)

$$\forall i, x : \|\nabla f_i(x)\| \leq G.$$

Remark 4.5. *The two assumptions above are not necessary to prove the results in this chapter. We prove this as a lemma in our follow up work, "Permutation-Based SGD: Is Random Optimal?" (Rajput et al., 2022), which is also presented as Lemma C.1 in Chapter 6.*

Assumption 4.6. (*Lipschitz Gradients*) The functions f_i are L -smooth, that is

$$\forall i, x, y : \|\nabla f_i(x) - \nabla f_i(y)\| \leq L\|x - y\|.$$

4.4 Optimal RR Convergence Rates for Quadratics

In this section, we will focus on strongly convex functions that are quadratic. We will provide a tight convergence rate that improves upon the existing rates and matches the $\Omega\left(\frac{1}{T^2} + \frac{n^2}{T^3}\right)$ lower bound by Safran and Shamir (2020) up to logarithmic factors.

For strongly convex functions that are a sum of smooth functions, Nagaraj et al. (2019) offer a rate of $\mathcal{O}\left(\frac{n}{T^2}\right)$, whereas for strongly convex quadratics Haochen and Sra (2019a) give a convergence rate of $\mathcal{O}\left(\frac{1}{T^2} + \frac{n^3}{T^3}\right)$. A closer comparison of these two rates reveals that neither of them can be tight due to the following observation. Assume that $n \ll K$. Then, that implies

$$\left(\frac{1}{T^2} + \frac{n^3}{T^3}\right) < \frac{n}{T^2}.$$

At the same time, if we assume that the number of data points is significantly larger than the number of epochs that we run RR for, *i.e.*, $n \gg K$ we have that

$$\left(\frac{1}{T^2} + \frac{n^3}{T^3}\right) > \frac{n}{T^2}.$$

In comparison, the known lower bound for quadratics given by Safran and Shamir (2020) is $\Omega\left(\frac{1}{T^2} + \frac{n^2}{T^3}\right)$. This makes one wonder what is the true convergence rate of RR in this case. We settle the optimal rates for quadratics here by providing an upper bound which, up to logarithmic factors, matches the best known lower bound.

For the special case of one dimensional quadratics, Safran and Shamir (2020) proved an upper bound matching the one we prove in this chapter. Further, the paper conjectures that the proof can be extended to the generic multidimensional case. However, the authors say that the main technical barrier for this extension is that it requires a special case of a matrix-valued arithmetic-geometric mean inequality, which has only been conjectured to be true but not yet proven. The authors further conjecture that their proof can be extended to general smooth and strongly convex functions, which turns out to not be true, as we show in Corollary 4.13. On the other hand, we believe that our proof can be extended to the more general family of strongly convex functions, where the Hessian is Lipschitz, similar to the the way Haochen and Sra (2019a) extend their proof to that case.

In addition to Assumptions 1-5 above, here we also assume the following:

Assumption 4.7. F is a quadratic function

$$F(x) = \frac{1}{2}x^T Hx + b^T x + c,$$

where H is a positive semi-definite matrix.

Note that this assumption is a little more general than the assumption that F is a sum of quadratics. Also, note that this assumption, in combination with the assumptions on strong convexity and Lipschitz gradients implies bounds on the minimum and maximum eigenvalues of the Hessian of F , that is,

$$\mu I \preceq H \preceq LI,$$

where I is the identity matrix and $A \preceq B$ means that $x^T(A - B)x \leq 0$ for all x .

Theorem 4.8. Under Assumptions 4.1-4.7, let the step size of RR be

$$\alpha = \frac{8 \log T}{T\mu}$$

and the number of epochs be

$$K \geq 128 \frac{L^2}{\mu^2} \log T.$$

Then, after T iterations RR achieves the following rate

$$\mathbb{E}[\|x_T - x^*\|^2] = \tilde{O}\left(\frac{1}{T^2} + \frac{n^2}{T^3}\right),$$

where $\tilde{O}(\cdot)$ hides logarithmic factors.

The exact upper bound and the full proof of this Theorem are given in Appendix A.1, but we give a proof sketch in the next subsection.

At this point, we would like to remark that the bound on epochs $K \geq 128 \frac{L^2}{\mu^2} \log T$ may be a bit surprising as K and T are dependent. However, note that since $T = nK$, we can show that the bound on K above is satisfied if we set the number of epochs to be greater than $C \log n$ for some constant C . Furthermore, we note that the dependence of K on $\frac{L}{\mu}$ (i.e., the condition number of F) is most probably not optimal. In particular both (Nagaraj et al., 2019) and (Haochen and Sra, 2019a) have a better dependence on the condition number.

The proof for Theorem 4.8 uses ideas from the works of Haochen and Sra (2019a) and Nagaraj et al. (2019). In particular, one of the central ideas in these two papers is that they aim to quantify the amount of progress made by RR over a single epoch. Both analyses decompose the progress of the iterates in an epoch as n steps of full gradient descent plus some noise term.

Similar to (Haochen and Sra, 2019a), we use the fact that the Hessian H of F is constant, which helps us better estimate the value of gradients around the minimizer. In contrast to that work, we do not require all individual components f_i to be quadratic, but rather the entire F to be a quadratic function.

An important result proved by (Nagaraj et al., 2019) is that during an epoch, the iterates do not steer off too far away from the starting point of the epoch. This allows one to obtain a reasonably good bound on the noise term, when one tries to approximate the stochastic gradient with the true gradient of F . In our analysis, we prove a slightly different version of the same result using an iterate coupling argument similar to the one in (Nagaraj et al., 2019).

The analysis of (Nagaraj et al., 2019) relies on computing the Wasserstein distance between the unconditional distribution of iterates and the distribution of iterates given a function sampled during an iteration. In our analysis, we use the same coupling, but we bypass the Wasserstein framework that (Nagaraj et al., 2019) suggests and directly obtain a bound on how far the coupled iterates move away from each other during the course of an epoch. This results, in our view, to a somewhat simpler and shorter proof.

4.4.1 Sketch of proof for Theorem 4.8

Now we give an overview of the proof. As mentioned before, similar to the previous works, the key idea is to perform a tight analysis of the progress made during an epoch. This is captured by the following Lemma.

Lemma 4.9. *Let the RR step size be $\alpha = \frac{4l \log T}{T\mu}$ and the total number of epochs be $K \geq 128 \frac{L^2}{\mu^2} \log T$, where $l \leq 2$. Then for any epoch,*

$$\mathbb{E} [\|x_0^j - x^*\|^2] \leq \left(1 - \frac{n\alpha\mu}{4}\right) \|x_0^{j-1} - x^*\|^2 + 16n\alpha^3 G^2 L^2 \mu^{-1} + 20n^3 \alpha^4 G^2 L^2. \quad (4.1)$$

Given the result in Lemma 4.9, proving Theorem 4.8 is a simple exercise. To do so, we simply unroll the recursion (4.1) for K consecutive epochs. For ease of notation, define $C_1 := 16G^2 L^2 \mu^{-1}$ and $C_2 := 20G^2 L^2$. Then,

$$\begin{aligned} \mathbb{E} [\|x_n^K - x^*\|^2] &\leq \left(1 - \frac{n\alpha\mu}{4}\right) \mathbb{E} [\|x_0^K - x^*\|^2] + C_1 n \alpha^3 + C_2 n^3 \alpha^4 \\ &\leq \left(1 - \frac{n\alpha\mu}{4}\right)^2 \mathbb{E} [\|x_0^{K-1} - x^*\|^2] + (C_1 n \alpha^3 + C_2 n^3 \alpha^4) \left(1 + \left(1 - \frac{n\alpha\mu}{4}\right)\right) \\ &\quad \vdots \\ &\leq \left(1 - \frac{n\alpha\mu}{4}\right)^{K+1} \mathbb{E} [\|x_0^0 - x^*\|^2] + (C_1 n \alpha^3 + C_2 n^3 \alpha^4) \sum_{j=1}^K \left(1 - \frac{n\alpha\mu}{4}\right)^{j-1} \\ &= \left(1 - \frac{n\alpha\mu}{4}\right)^{K+1} \|x_0^0 - x^*\|^2 + (C_1 n \alpha^3 + C_2 n^3 \alpha^4) \sum_{j=1}^K \left(1 - \frac{n\alpha\mu}{4}\right)^{j-1}. \end{aligned}$$

We can now use the fact that $(1 - x) \leq e^{-x}$ and $\left(1 - \frac{n\alpha\mu}{4}\right) \leq 1$, to get the following bound:

$$\mathbb{E} [\|x_n^K - x^*\|^2] \leq e^{-\frac{n\alpha\mu}{4}K} \|x_0^0 - x^*\|^2 + (C_1 n \alpha^3 + C_2 n^3 \alpha^4)K.$$

By setting the step size to be $\alpha = \frac{4l \log T}{T\mu}$ and noting that $T = nK$, we get that

$$\mathbb{E} [\|x_n^K - x^*\|^2] \leq e^{-n \frac{4l \log T}{T\mu} \frac{\mu}{4} K} \|x_0^0 - x^*\|^2 + (n \alpha^3 C_1 + \alpha^4 n^3 C_2)K$$

$$\begin{aligned}
&= e^{-l \log T} \|x_0^0 - x^*\|^2 + \tilde{\mathcal{O}} \left(\frac{1}{T^2} + \frac{n^2}{T^3} \right) \\
&= \frac{\|x_0^0 - x^*\|^2}{T^l} + \tilde{\mathcal{O}} \left(\frac{1}{T^2} + \frac{n^2}{T^3} \right).
\end{aligned}$$

Noting that $\|x_0^0 - x^*\| \leq D$ and choosing $l = 2$ gives us the result of Theorem 4.8.

4.4.2 With- and without-replacement stochastic gradients are close

One of the key lemmas in (Nagaraj et al., 2019) establishes that once RR iterates get close enough to the global minimizer x^* , then any iterate at any time during an epoch x_i^j stays close to the iterate at the beginning of that epoch. To be more precise, the lemma we refer to is the following.

Lemma 4.10. [Nagaraj et al. (2019, Lemma 5)] *Under the assumptions of Theorem 4.8,*

$$\mathbb{E}[\|x_i^j - x_0^j\|^2] \leq 5i\alpha^2 G^2 + 2i\alpha(F(x_0^j) - F(x^*)).$$

We would like to note that Lemma 4.10 is slightly different from the one in (Nagaraj et al., 2019), which instead uses $\mathbb{E}[F(x_i^j) - F(x^*)]$ rather than $(F(x_i^j) - F(x^*))$, but their proof can be adapted to obtain the version written above. For the formal version of Lemma 4.10, please see Lemma A.6 in the Appendix.

Now, consider the case when the iterates are very close to the optimum and hence $F(x_i^j) - F(x^*) \approx 0$. Then, Lemma 4.10 implies that $\mathbb{E}[\|x_i^j - x_0^j\|^2]$ does not grow quadratically in i which would generically happen for i gradient steps, but it rather grows linearly in i . This is an important and useful fact for RR: it shows that all iterates within an epoch remain close to x_0^j .

Hence, since the iterates of RR do not move too much during an epoch, then the gradients computed throughout the epoch at points x_i^j should be well approximated by gradients computed on the x_0^j iterate. Roughly, this translates to the following observation: the n

gradient steps taken through a single epoch are almost equal to n steps of full gradient descent computed at x_0^j . This is in essence what allows RR to achieve better convergence than SGD - an epoch can be approximated by n steps of gradient descent.

Now, let σ^j represent the random permutation of the n functions f_i during the j -th epoch. Thus, $\sigma^j(i)$ is the index of the function chosen at the i -th iteration of the j -th epoch. Proving Lemma 4.10 requires proving that the function value of $f_{\sigma^j(i)}(x_i^j)$, in expectation, is almost equal to $F(x_i^j)$. In particular, we prove the following claim in our supplemental material.

Claim 4.11. [Nagaraj et al. (2019, Lemma 4)] *If $\alpha \leq \frac{2}{L}$, then for any epoch j and i -th ordered iterate during that epoch*

$$\left| \mathbb{E} \left[F(x_i^j) - f_{\sigma^j(i)}(x_i^j) \mid x_0^j \right] \right| \leq 2\alpha G^2. \quad (4.2)$$

This claim establishes that RR behaves almost like SGD, for which the following is true: $\mathbb{E}[f_{\sigma^j(i)}(x_i^j)] = \mathbb{E}[F(x_i^j)]$. To prove this claim, (Nagaraj et al., 2019) consider the conditional distribution of iterates, given the current function index, that is $x_i^j | \sigma_i(j)$, and the unconditional distribution of the iterates x_i^j . Then, they prove that the absolute difference $|\mathbb{E}[F(x_i^j)] - \mathbb{E}[f_{\sigma^j(i)}(x_i^j)]|$ can be upper bounded by the Wasserstein distance between these two distributions. To further upper bound the Wasserstein distance, they propose a coupling between the two distributions. To prove our slightly different version of Lemma 4.10, we proved (4.2) without using this Wasserstein framework. Instead, we use the same coupling argument to directly get a bound on (4.2). Below we explain the coupling and provide a short intuition.

Consider the conditional distribution of $\sigma^j | \sigma^j(i) = s$. If we take the distribution of $\sigma | \sigma(i) = 1$, we can generate the support of $\sigma^j | \sigma^j(i) = s$ by taking all permutations $\sigma | \sigma(i) = 1$ and by swapping 1 and s among them. This is essentially a coupling between these two distributions, proposed in (Nagaraj et al., 2019). Now, if we use this coupling to convert a permutation in $\sigma | \sigma(i) = 1$ to a permutation $\sigma | \sigma(i) = s$, the corresponding $x_i | \sigma(i) = 1$ and

$x_i | \sigma(i) = s$ would be within a distance of $2\alpha G$. This distance bound is Lemma 2 of (Nagaraj et al., 2019).

We can now use such distance bound, and let $v_{(1,s)}$ denote a (random) vector whose norm is less than $2\alpha G$. Then,

$$\begin{aligned}
 \mathbb{E} [f_{\sigma(i)}(x_i)] &= \frac{1}{n} \sum_{s=1}^n \mathbb{E} [f_{\sigma(i)}(x_i) | \sigma(i) = s] \\
 &= \frac{1}{n} \sum_{s=1}^n \mathbb{E} [f_s(x_i) | \sigma(i) = s] \\
 &= \frac{1}{n} \sum_{s=1}^n \mathbb{E} [f_s(x_i + v_{(1,s)}) | \sigma(i) = 1] \\
 &\leq \frac{1}{n} \sum_{s=1}^n \mathbb{E} [f_s(x_i) + (2\alpha G^2) | \sigma(i) = 1] \\
 &= \mathbb{E} [F(x_i) | \sigma(i) = 1] + 2\alpha G^2.
 \end{aligned}$$

Similarly, for any $s \in \{1, \dots, n\}$:

$$\mathbb{E} [f_{\sigma(i)}(x_i)] \leq \mathbb{E} [F(x_i) | \sigma(i) = s] + 2\alpha G^2.$$

Therefore,

$$\begin{aligned}
 \mathbb{E} [f_{\sigma(i)}(x_i)] &\leq \frac{1}{n} \sum_{s=1}^n \mathbb{E} [F(x_i) | \sigma(i) = s] + 2\alpha G^2 \\
 &\leq \mathbb{E} [F(x_i)] + 2\alpha G^2.
 \end{aligned}$$

Similarly, we can prove that

$$\mathbb{E} [f_{\sigma(i)}(x_i)] \geq \mathbb{E} [F(x_i)] - 2\alpha G^2.$$

Combining these two results we obtain (4.2). The detailed proof of Claim 4.11 is provided in the appendix.

The full proof of Theorem 1 requires some more nuanced bounding derivations, and the complete details can be found in Appendix A.1.

4.5 Lower Bound for General Case

In the previous section, we establish that for quadratic functions the $\Omega\left(\frac{1}{T^2} + \frac{n^2}{T^3}\right)$ lower-bound by Safran and Shamir (2020) is essentially tight. This still leaves open the possibility that a tighter lower bound may exist for strongly convex functions that are not quadratic. After all, the best convergence rate known for strongly convex functions that are sums of smooth functions is of the order of n/T^2 .

Indeed, in this section, we show that the convergence rate of $\mathcal{O}\left(\frac{n}{T^2}\right)$ established by Nagaraj et al. (2019) is tight.

For a certain constant C (see Appendix A.2 for the formal version of the theorem), we show the following theorem

Theorem 4.12. *There exists a strongly convex function F that is the sum of n smooth convex functions, such that for any step size*

$$\frac{1}{T} \leq \alpha \leq \frac{C}{n},$$

the error after T total iterations of RR satisfies

$$\mathbb{E}[\|x_T - x^*\|^2] = \Omega\left(\frac{n}{T^2}\right).$$

The full proof of this theorem is provided in Appendix A.2, but we give an intuitive explanation of the proof later in this section.

Note that the theorem above establishes the existence of a function for which RR converges at rate $\Omega(\frac{n}{T^2})$, but only for the step size range $\frac{1}{T} \leq \alpha \leq \frac{C}{n}$. This is the range of the most interest because most of the upper bounds and convergence guarantees of RR (and SGD) work in this step size range. However, it would still be desirable to get a function on

which RR converges at rate $\Omega(n/T^2)$ for all step sizes. Such a function would be difficult to optimize, no matter how much we tune the step size. Indeed, we show that based on Theorem 4.12, we can create such a function. To do that, we use a function proposed by Safran and Shamir (2020, Proposition 1), which converges slowly outside of the step size range $\frac{1}{T} \leq \alpha \leq \frac{C}{n}$.

Safran and Shamir (2020) show that there exists a strongly convex function F_2 , which is the sum of n quadratics, such that for step size $\alpha \leq \frac{1}{T}$, the expected error satisfies $\mathbb{E}[\|x_T - x^*\|^2] = \Omega(1)$ (see the proof of Proposition 1, pg. 10-12 in their paper). Further, for the same function F_2 , the proof of that proposition can be adapted directly to get $\mathbb{E}[\|x_T - x^*\|^2] = \Omega\left(\frac{1}{n}\right)$ for any step size $\alpha \geq \frac{C}{n}$, for any constant C .

Using this function F_2 and the function F from Theorem 4.12, we can create a function on which RR converges at rate $\Omega\left(\frac{n}{T^2}\right)$ for all step sizes.

Corollary 4.13. *There exists a 2-Dimensional strongly convex function that is the sum of n smooth convex functions, such that for any $\alpha > 0$*

$$\mathbb{E}[\|x_T - x^*\|^2] = \Omega\left(\frac{n}{T^2}\right).$$

The proof of this corollary is provided in Appendix A.3.

Thus overall, we get that for any fixed step size $\mathbb{E}[\|x_T - x^*\|^2] = \Omega\left(\frac{n}{T^2}\right)$. Next, we try to explain the function construction and proof technique behind Theorem 4.12. The construction of the lower bound is similar to the one used by Safran and Shamir (2020). The difference is that the prior work considers quadratic functions, while we consider a slightly modified piece-wise quadratic function.

Specifically, we construct the following function $F(x) = \frac{1}{n} \sum_{i=1}^n f_i(x)$ as

$$F(x) = \begin{cases} \frac{x^2}{2}, & \text{if } x \geq 0 \\ \frac{Lx^2}{2}, & \text{if } x < 0, \end{cases}$$

where n is an even number. Of the n component functions f_i , half of them are defined as follows:

$$\text{if } i \leq \frac{n}{2}, \text{ then } f_i(x) = \begin{cases} \frac{x^2}{2} + \frac{Gx}{2}, & \text{if } x \geq 0 \\ \frac{Lx^2}{2} + \frac{Gx}{2}, & \text{if } x < 0, \end{cases}$$

and the other half of the functions are defined as follows:

$$\text{if } i > \frac{n}{2}, \text{ then } f_i(x) = \begin{cases} \frac{x^2}{2} - \frac{Gx}{2}, & \text{if } x \geq 0 \\ \frac{Lx^2}{2} - \frac{Gx}{2}, & \text{if } x < 0. \end{cases}$$

For our construction, we set L to be a big enough positive constant. See for example, Fig. 4.1.

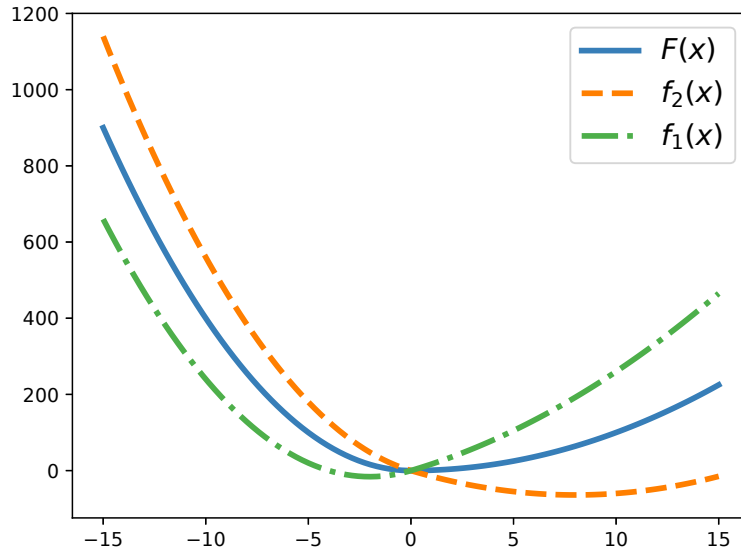


Figure 4.1: Lower bound construction. Note that $f_1(x)$ represents the component functions of the first kind, and $f_2(x)$ represents the component functions of the second kind, and $F(x)$ represents the overall function.

Next we ought to verify that this function abides to Assumptions 4.1-4.6. Note that Assumption 4.1 is satisfied, as it can be seen that functions f_i 's are all continuous and convex. Next, we need to show that Assumption 4.2 holds, that is F is strongly convex.

We will show that this is true by proving the following equivalent definition of strong convexity: a function f is μ -strongly convex if $g(x) := f(x) - \frac{\mu}{2}\|x\|^2$ is convex. We can see that this is true for F with $\mu = 1$.

In the proof of Theorem 4.12, we initialize at the origin. In that case, in the proof we also prove that Assumptions 4.3 and 4.4 hold. In particular, we show that the iterates do not go outside of a bounded domain, and inside this domain, the gradient is bounded by G . Finally, let us focus on Assumption 4.6. To prove that these functions have Lipschitz gradients, we need to show

$$\forall x, y : |\nabla f_i(x) - \nabla f_i(y)| \leq L|x - y|.$$

If $xy \geq 0$, that is x and y lie on the same side of the origin, then this is simple to see because they both lie on the same quadratic. Otherwise WLOG, assume $x < 0$ and $y > 0$. Also, assume WLOG that f_i is function of the first kind, that is $i \leq \frac{n}{2}$ and hence the linear term in $f_i(x)$ is $\frac{Gx}{2}$. Then,

$$\begin{aligned} |\nabla f_i(x) - \nabla f_i(y)| &= \left| Lx + \frac{G}{2} - y - \frac{G}{2} \right| \\ &= y - Lx \\ &\leq Ly - Lx \\ &\leq L|y - x|. \end{aligned}$$

Overall, the difficulty in the analysis comes from the fact that unlike the functions considered by Safran and Shamir (2020), our functions are piece-wise quadratics.

Let us initialize at $x_0^1 = 0$ (the minimizer). We will show that in expectation, at the end of K epochs, the iterate would be at a certain distance (in expectation). Note that the progress made over an epoch is just the sum of gradients (multiplied by $-\alpha$) over the

epoch:

$$x_n^j - x_0^j = -\alpha \sum_{i=1}^n \nabla f_{\sigma^j(i)}(x_i)$$

where $\sigma^j(i)$ represents the index of the i -th function chosen in the j -th epoch. Next, note that the gradients from the linear components $\pm \frac{G}{2}x$ are equal to $\pm \frac{G}{2}$, that is they are constant. Thus, they will cancel out over an epoch.

However the gradients from the quadratic components do not cancel out, and in fact that part of the gradient will not even be unbiased, in the sense that if $x_t \geq 0$, the gradient at x_t from the quadratic component $\frac{x^2}{2}$ will be less in magnitude than the gradient from the quadratic component $\frac{Lx^2}{2}$ at $-x_t$.

The idea is to now ensure that if an epoch starts off near the minimizer, then the iterates spend a certain amount of time in the $x < 0$ region, so that they “accumulate” a lot of gradients of the form Lx , which makes the sum of the gradients at the end of the epoch biased away from the minimizer.

To ensure that the iterates spend some time in the $x < 0$ region, we analyze the contribution of the linear components during the epoch. This is because when the iterates are already near the minimizer $x \approx 0$, the gradient contribution of the quadratic terms would be small, and the dominating component during an epoch would come from the linear terms. What this means is that in the middle of an epoch, it is the linear terms which contribute the most towards the “iterate movement”, even though at the end of that epoch their gradients get cancelled out and what remains is the contribution of the quadratic terms.

Then, to obtain a lower bound matching the upper bound given by Nagaraj et al. (2019), observe that it is indeed this contribution of the linear terms that we require to get a tight bound on. This is because, the upper bound from the aforementioned work was also in fact directly dependent on the movement of iterates away from the minimizer during an epoch, caused by the stochasticity in the gradients (cf. Lemma 5 of Nagaraj et al. (2019)). We give below the informal version of the main lemma for the proof:

Lemma 4.14. [Informal] *Let $(\sigma_1, \dots, \sigma_n)$ be a random permutation of the following:*

$$\underbrace{(+1, \dots, +1)}_{\frac{n}{2} \text{ times}}, \underbrace{(-1, \dots, -1)}_{\frac{n}{2} \text{ times}}.$$

Then for $i < n/2$, $\mathbb{E}[\sum_{j=1}^i \sigma_j] \geq C\sqrt{i}$, where C is a universal constant.

Please see Lemma A.17 in Appendix A.2 for the formal version of this lemma.

For the purpose of intuition, ignore the contribution of gradients from the quadratic terms. Then, the lemma above says that during an epoch, the gradients from the linear terms would move the iterates approximately $\Omega\left(\alpha\sqrt{n}\frac{G}{2}\right)$ away from the minimizer (after we multiply by the step size α).

This implies that in the middle of an epoch, with (almost) probability $1/2$ the iterates would be near $x \approx -\Omega\left(\alpha\sqrt{n}\frac{G}{2}\right)$ and with (almost) probability $1/2$ the iterates would be near $x \approx \Omega\left(\alpha\sqrt{n}\frac{G}{2}\right)$. Hence, over the epoch, the accumulated quadratic gradients multiplied by the step size would look like

$$\begin{aligned} \sum_{i=1}^n \mathbb{E}\left[-\alpha(L\mathbb{1}_{x_i^j < 0} + \mathbb{1}_{x_i^j \geq 0})x_i^j\right] &\approx -\alpha \sum_{i=1}^n \left(\frac{1}{2}L\Omega\left(-\alpha\sqrt{n}\frac{G}{2}\right) + \frac{1}{2}\Omega\left(\alpha\sqrt{n}\frac{G}{2}\right)\right) \\ &= \Omega(L\alpha^2 n \sqrt{n}). \end{aligned}$$

If this happens for K epochs, we get that the accumulated error would be

$\Omega(L\alpha^2 n \sqrt{n} K) = \Omega\left(\frac{1}{\sqrt{n}K}\right)$ for $\alpha \in [1/nK, 1/n]$. Since $E[|x_T|] \geq 1/\sqrt{n}K$, we know that $E[|x_T - 0|^2] \geq 1/nK^2 = n/T^2$. Since 0 is the minimizer of our function in this setting, we have constructed a case where RR achieves error

$$E[|x_T - x^*|^2] \geq n/T^2.$$

This completes the sketch of the proof and the complete proof of Theorem 4.12 is given in Appendix A.2.

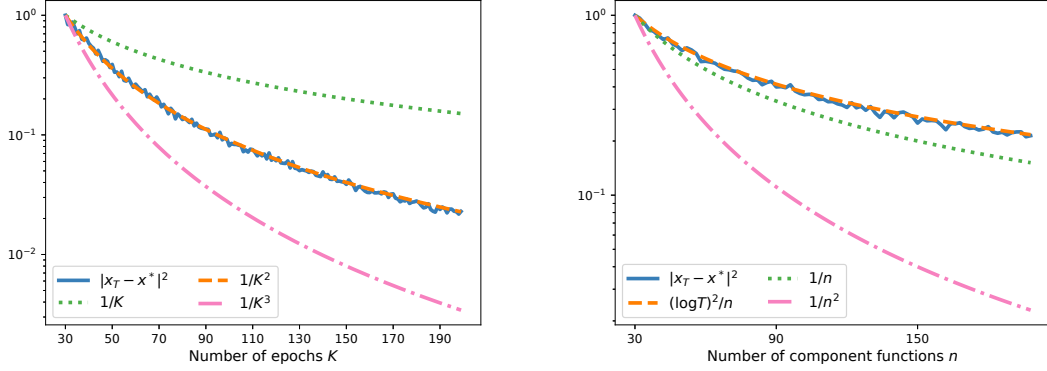


Figure 4.2: Running RR on the function F used in our lower bound (Theorem 4.12) confirms that the rate of convergence of RR on this function is indeed $\Omega(\frac{1}{nK^2}) = \Omega(\frac{n}{T^2})$. The curves are normalized so that they begin at the same point.

4.5.1 Numerical verification

To verify our lower bound of Theorem 4.12, we ran RR on the function described in Eq. (4.5) with $L = 4$.¹ The step size regimes that were considered were $\alpha = \frac{1}{T}, \frac{2 \log T}{T}, \frac{4 \log T}{T}, \frac{8 \log T}{T}$, and $\frac{1}{n}$. The plot for $\alpha = \frac{4 \log T}{T}$ is shown in Figure 4.2. The plots for the other step size regimes are provided in Appendix A.4.

The step size regimes considered cover the range specified in the statement of Theorem 4.12. Looking at Figure 4.2 (and the figures in Appendix A.4 for the other step size regimes), the dependence of the convergence rate on K indeed looks exactly like $1/K^2$. However, looking at the figures for the dependence of the convergence rate on n , we see that they look like $\frac{(\log T)^2}{n}$. This suggests that the tightest possible lower bound for RR with constant step size on strongly convex smooth functions might have a logarithmic term in the numerator. Next, we explain the details of the experiment.

Consider any one of the step size regimes specified above, say $\alpha = \frac{4 \log T}{T}$. For this regime, we ran two experiments:

1. We fix $n = 500$ and vary K from 30 to 200, and

¹The code for these experiments is available at <https://github.com/shashankrajput/RR>.

2. we fix $K = 500$ and vary n from 30 to 200.

Consider the first experiment, where $n = 500$ and K is varied. For each value of K , say $K = 50$, we set $\alpha = \frac{4 \log T}{T} = \frac{4 \log(nK)}{nK} = \frac{4 \log(500*50)}{500*50}$ and ran RR with this constant step size α on the sum of $n = 500$ functions for $K = 50$ epochs, and the final error was recorded. This was repeated 1000 times to reduce variance. The final mean error after these 1000 runs gave us one point, which we plotted for $K = 50$ on the top subfigure of Figure 4.2. Repeating the same for all values of K from 30 to 200 gave us the top subfigure of Figure 4.2. The same procedure was followed for the second experiment where we fix K and vary n , and that gave us the bottom subfigure of Figure 4.2. The optimization was initialized at the origin, that is $x_0^1 = 0$. These pairs of experiments were performed for all values of step size regimes in the list $(\frac{1}{T}, \frac{2 \log T}{T}, \frac{4 \log T}{T}, \frac{8 \log T}{T}, \text{ and } \frac{1}{n})$.

Now, we justify the ranges of n and K considered in our experiments. We wanted to verify that the lower bound on the error of RR is indeed

$$\Omega\left(\frac{n}{T^2}\right) = \Omega\left(\frac{1}{nK^2}\right) \quad [\text{Theorem 4.12}]$$

instead of the previously known best lower bound

$$\Omega\left(\frac{1}{T^2} + \frac{n^2}{T^3}\right) = \Omega\left(\frac{1}{nK^2} \left(\frac{1}{n} + \frac{1}{K}\right)\right). \quad [\text{Safran and Shamir (2020)}]$$

Looking at the RHS of the two equations above, we can see that the dependence of the two lower bounds on K differs only when $n \gg K$ and the dependence on n differs only when $K \gg n$. Thus for example, when we wanted to check dependence on K , we set $n = 500$ which was bigger than every K in the range 30 to 200.

4.5.2 Discussion on possible improvements

Theorem 4.12 hints that for faster convergence rates in the epoch based random shuffling SGD, we would not just require smooth and strongly convex functions, but also potentially require that the Hessians of such functions to be Lipschitz.

We conjecture that Hessian Lipschitzness is sufficient to get the convergence rate of Theorem 4.8. We think that this is interesting, because the optimal rates for both SGD and GD only require strong convexity and gradient smoothness. However, here we prove that an optimal rate for RR requires the function to be quadratic as well (or at the very least have a Lipschitz Hessian), and RR seems to converge slower if the Hessian is not Lipschitz.

4.6 Conclusions and Future Work

Random Reshuffling has long puzzled researchers. From a practical point of view, it always seems to outperform SGD, and is the algorithm of choice for training modern machine learning models. From a theoretical point of view, RR has resisted tight convergence analysis that establish its performance benefits. A recent wave of work established that indeed RR can be faster than SGD sampling, however a gap still remained between the achievable rates and the best known lower bounds.

In this chapter we settle the optimal performance of Random Reshuffling for functions that are quadratics, and strongly convex functions that are sums of n smooth functions. Our results indicate that a possible improvement in convergence rates may require a fundamentally different step size rule and significantly different function assumptions.

As future directions, we believe that it would be interesting to establish rates for variants of RR that do not re-permute the functions at every epoch. This is something that is common in practice, where a random permutation is only performed once every few epochs without a significant drop in performance. Current theoretical bounds are inadequate to explain this phenomenon, and a new theoretical breakthrough may be required to tackle it.

We however believe that one of the strongest new theoretical insights introduced by Nagaraj et al. (2019) and used in our analyses can be of significance in a potential attempt to analyze other variants of RR as the one above. This insight is that of iterate coupling. That is the property that RR iterates are only mildly perturbed after swapping only two elements of a permutation. Such a property is reminiscent to that of algorithmic stability, and a deeper connection between that and iterate coupling is left as a meaningful intellectual endeavor for future work.

5 RANDOM RESHUFFLING IN DISTRIBUTED AND FEDERATED LEARNING

In distributed learning, **local SGD** (also known as federated averaging) and its simple baseline **minibatch SGD** are widely studied optimization methods. Most existing analyses of these methods assume independent and unbiased gradient estimates obtained via *with-replacement* sampling. In contrast, we study *shuffling-based* variants: **minibatch** and **local Random Reshuffling**, which draw stochastic gradients without replacement and are thus closer to practice. For smooth functions satisfying the Polyak-Łojasiewicz condition, we obtain convergence bounds (in the large epoch regime) which show that these shuffling-based variants converge *faster* than their with-replacement counterparts. Moreover, we prove matching lower bounds showing that our convergence analysis is *tight*. Finally, we propose an algorithmic modification called **synchronized shuffling** that leads to convergence rates *faster* than our lower bounds in near-homogeneous settings.

5.1 Introduction

Distributed learning within the framework of federated learning (Konečný et al., 2016; McMahan et al., 2017) has witnessed increasing interest recently. A key property of this framework is that models are trained locally using only private data on devices/machines distributed across a network, while parameter updates are aggregated and synchronized at a server.¹ Communication is often the key bottleneck for federated learning, which drives the search for algorithms that can train fast while requiring less communication—see Li et al. (2020a); Kairouz et al. (2021) for recent surveys.

This chapter presents our paper "Minibatch vs Local SGD with Shuffling: Tight Convergence Bounds and Beyond" (Yun et al., 2022).

¹A distinctive feature of federated learning is that not all devices necessarily participate in the updates; however, we focus on the full participation setting in this paper.

A basic algorithm for federated learning is local stochastic gradient descent (SGD), also known as federated averaging. The goal is to minimize the global objective that is an average of the local objectives. In **local SGD**, we have M machines and a server. After each round of communication, each of the M machines locally runs B steps of SGD on its local objective. Every B iterations, the server aggregates the updated local iterates from the machines, averages them, and then synchronizes the machines with the average. Convergence analysis of local SGD and its variants has drawn great interest recently (Dieuleveut and Patel, 2019; Haddadpour et al., 2019; Haddadpour and Mahdavi, 2019; Stich, 2019; Yu et al., 2019; Li et al., 2020b,c; Koloskova et al., 2020; Khaled et al., 2020; Spiridonoff et al., 2020; Karimireddy et al., 2020; Stich and Karimireddy, 2020; Qu et al., 2020).

Of the many, the biggest motivation for our paper comes from the line of work by Woodworth et al. (2020a,b, 2021). In (Woodworth et al., 2020a,b), **minibatch SGD** is studied as a simple yet powerful baseline for this intermittent communication setting. Instead of locally updating the iterates B times, minibatch SGD aggregates B gradients (evaluated at the last synced iterate) from each of the M machines, forms a minibatch of size MB , and then updates the shared iterate. Given the same M and B , local SGD and minibatch SGD have the same number of gradient computations per round of communication, so it is worthwhile to understand which converges faster. Woodworth et al. (2020a,b) point out that many existing analyses on local SGD show inferior convergence rate compared to minibatch SGD. Through their new upper and lower bounds, they identify regimes where local SGD can be faster than minibatch SGD.

While the theory of local and minibatch SGD has seen recent progress, there is still a gap between what is analyzed versus what is actually used. Most theoretical results assume independent and unbiased gradient estimates obtained via *with-replacement* sampling of stochastic gradients (i.e., choosing training data indices uniformly at random). In contrast, most practitioners use *without-replacement* sampling, where they shuffle indices randomly and access them sequentially.

Convergence analysis of without-replacement methods is challenging because gradients sampled within an epoch lack independence. As a result, the standard theory based on independent gradient estimates does not apply to shuffling-based methods. While shuffling-based methods are believed to be faster in practice (Bottou, 2009), broad theoretical understanding of such methods remains elusive, except for noteworthy recent progress mainly focusing on the analysis of SGD (Gürbüzbalaban et al., 2015; Haochen and Sra, 2019b; Nagaraj et al., 2019; Nguyen et al., 2020; Safran and Shamir, 2020, 2021; Rajput et al., 2020, 2022; Ahn et al., 2020; Mishchenko et al., 2020, 2021; Tran et al., 2021). These results indicate that in the *large-epoch regime* (where the number of epochs is greater than some threshold), without-replacement SGD converges faster than with-replacement SGD.

5.1.1 Our contributions

We analyze convergence rates of without-replacement versions of local and minibatch SGD, where local component functions are reshuffled at every epoch. We call the respective algorithms **local RR** (Algorithm 5.1) and **minibatch RR** (Algorithm 5.2), and their with-replacement counterparts **local SGD** and **minibatch SGD**. Our key contributions are as follows:

- In Section 5.3, we present convergence bounds on minibatch and local RR for L -smooth functions satisfying the μ -Polyak-Łojasiewicz condition (Theorems 5.6 & 5.7). Our theorems give *high-probability* bounds, a departure from the common in-expectation bounds in the literature. We show that minibatch and local RR converge *faster* than minibatch and local SGD when the number of epochs is sufficiently large. We also identify a regime where local RR converges as fast as minibatch RR: when synchronization happens frequently enough and local objectives are not too heterogeneous. See also Appendix B.1 for a detailed comparison with existing upper bounds.
- In Section 5.4, we prove that the upper bounds obtained in Section 5.3 are *tight*, in all

factors except L and μ . We present Theorems 5.8 & 5.10 and Proposition 5.11 which show lower bounds that match the upper bound up to a factor of L^2/μ^2 . Our lower bound on local RR indicates that if the synchronization interval B is too large, then local RR has *no gain* from parallel computation.

- In Section 5.5, we propose a simple modification called synchronized shuffling that allows us to *bypass* the lower bounds in Section 5.4, at the cost of a slight increase in communication. By having the server broadcast random permutations to local machines, we show that in near-homogeneous settings, the modified algorithms converge faster than the lower bounds (Theorems 5.13 & 5.14).
- In Appendix B.3, we present numerical experiments that corroborate our theoretical findings.

5.2 Problem setup

Notation. For a natural number $a \in \mathbb{N}$, let $[a] := \{1, 2, \dots, a\}$. Let $\mathcal{S}_a$ be the set of all permutations of $[a]$. Since our indices start from 1, we redefine the modulo operation between $a \in \mathbb{Z}$ and $b \in \mathbb{N}$ as $a \bmod b := a - \lfloor \frac{a-1}{b} \rfloor b$, to make $a \bmod b \in [b]$.

Optimization task. Consider M machines, each with its objective $F^m(\mathbf{x}) := \frac{1}{N} \sum_{i=1}^N f_i^m(\mathbf{x})$, for $m \in [M]$. The m -th machine has access only to the gradients of its own N local components $f_1^m(\mathbf{x}), \dots, f_N^m(\mathbf{x})$. In this setting, we wish to minimize the global objective function which is an average of the local objectives: $F(\mathbf{x}) := \frac{1}{M} \sum_{m=1}^M F^m(\mathbf{x}) = \frac{1}{MN} \sum_{m=1}^M \sum_{i=1}^N f_i^m(\mathbf{x})$.

Further, we assume that each individual component function f_i^m is L -smooth, so that

$$f_i^m(\mathbf{y}) \leq f_i^m(\mathbf{x}) + \langle \nabla f_i^m(\mathbf{x}), \mathbf{y} - \mathbf{x} \rangle + \frac{L}{2} \|\mathbf{y} - \mathbf{x}\|^2, \text{ for all } \mathbf{x}, \mathbf{y} \in \mathbb{R}^d, \quad (5.1)$$

and that the global objective F satisfies the μ -Polyak-Łojasiewicz (PL) condition.²

$$\frac{1}{2} \|\nabla F(\mathbf{x})\|^2 \geq \mu(F(\mathbf{x}) - F^*) \text{ for all } \mathbf{x} \in \mathbb{R}^d, \quad \text{where } \mu > 0. \quad (5.2)$$

Algorithms. Under the above setting, we analyze local RR (Algorithm 5.1) and minibatch RR (Algorithm 5.2) and characterize their worst-case convergence rates.³ The algorithms are run over K epochs, i.e., K passes over the entire component functions. At the beginning of epoch k , each machine m shuffles its local component functions $\{f_i^m\}_{i=1}^N$ using a random permutation $\sigma_k^m \sim \text{Unif}(\mathcal{S}_N)$. In local RR, each machine makes B local RR updates to its iterate by sequentially accessing its shuffled component functions, before the server aggregates iterates from all the machines and then synchronizes the machines with the average iterate. In minibatch RR, instead of making B local updates, each machine collects B gradients evaluated at the last iterate, and the server aggregates them to make an update using these MB gradients. Since these two algorithms use the same amount of communication and local gradients, minibatch RR is a simple yet powerful baseline for local RR.

Below, we collect our assumptions on the algorithm parameters used throughout the paper.

Assumption 5.1 (Algorithm parameters). *We assume $M \geq 1$, $N \geq 2$, and $K \geq 1$. Also, assume that B divides N . We restrict $1 \leq B \leq \frac{N}{2}$ for minibatch RR because $B = N$ makes the algorithm equal to GD. We also assume $2 \leq B \leq N$ for local RR because $B = 1$ makes the two algorithms the same. We choose a constant step-size scheme, i.e., $\eta > 0$ is kept constant over all updates.*

We next state assumptions on intra- and inter-machine deviations used in this paper.⁴

²PL functions can be thought as a nonconvex generalization of strongly convex functions.

³In Algorithms 5.1 and 5.2, consider SYNC SHUF as FALSE for now. We will discuss SYNC SHUF in Section 5.5.

⁴Assumptions 5.2, 5.3 & 5.4 require that they hold for the whole $\mathbb{R}^d$. We discuss ways to avoid it in Appendix B.4.7.

Algorithm 5.1 Local RR (with and without SYNC SHUF)

Input: Initialization $\mathbf{y}_0$, step-size η , # machines M , # components N , # epochs K , sync interval B .

- 1: Initialize $\mathbf{x}_{1,0}^m := \mathbf{y}_0$ for all $m \in [M]$.
- 2: **for** $k \in [K]$ **do**
- 3: **if** SYNC SHUF = TRUE **then** ▷ Local RR with SYNC SHUF
- 4: Sample $\sigma \sim \text{Unif}(\mathcal{S}_N)$, $\pi \sim \text{Unif}(\mathcal{S}_M)$.
- 5: Set $\sigma_k^m(i) := \sigma((i + \frac{N}{M}\pi(m)) \bmod N)$ for all $m \in [M]$, $i \in [N]$.
- 6: **else** ▷ Local RR
- 7: Sample $\sigma_k^m \sim \text{Unif}(\mathcal{S}_N)$ independently and locally, for all $m \in [M]$.
- 8: **end if**
- 9: **for** $i \in [N]$ **do**
- 10: **for** $m \in [M]$ **do locally**
- 11: Update $\mathbf{x}_{k,i}^m := \mathbf{x}_{k,i-1}^m - \eta \nabla f_{\sigma_k^m(i)}^m(\mathbf{x}_{k,i-1}^m)$.
- 12: **end for**
- 13: **if** B divides i **then**
- 14: Aggregate and average $\mathbf{y}_{k,\frac{i}{B}} := \frac{1}{M} \sum_{m=1}^M \mathbf{x}_{k,i}^m$.
- 15: Synchronize $\mathbf{x}_{k,i}^m := \mathbf{y}_{k,\frac{i}{B}}$, for all $m \in [M]$.
- 16: **end if**
- 17: **end for**
- 18: $\mathbf{x}_{k+1,0}^m := \mathbf{y}_{k,\frac{N}{B}}$, for all $m \in [M]$.
- 19: **end for**
- 20: **return** the last iterate $\mathbf{y}_{K,\frac{N}{B}}$.

Assumption 5.2 (Intra-machine deviation). *There exists $\nu \geq 0$ such that for all $m \in [M]$ and $i \in [N]$,*

$$\|\nabla f_i^m(\mathbf{x}) - \nabla F^m(\mathbf{x})\| \leq \nu, \text{ for all } \mathbf{x} \in \mathbb{R}^d.$$

Assumption 5.2 requires that the difference between the gradient of each local component function $f_i^m(\mathbf{x})$ and its corresponding local objective function $F^m(\mathbf{x})$ is uniformly bounded. It models the variance of local components f_i^m *within* each machine. While the uniform boundedness requirement may look strong, we use this assumption to prove high-probability upper bounds, which are stronger than the common in-expectation bounds. See Appendix B.1 for comparisons with other assumptions, and also Appendix B.4.7 for ways to avoid uniform boundedness over the *entire* $\mathbb{R}^d$.

The next two assumptions capture the deviation *across* different machines, i.e., the degree of heterogeneity, in two different levels of granularity: *objective-wise* and *component-*

Algorithm 5.2 Minibatch RR (with and without SYNC SHUF)

Input: Initialization $\mathbf{x}_0$, step-size η , # machines M , # components N , # epochs K , sync interval B .

- 1: Initialize $\mathbf{x}_{1,0} := \mathbf{x}_0$.
- 2: **for** $k \in [K]$ **do**
- 3: **if** SYNC SHUF = TRUE **then** ▷ Minibatch RR with SYNC SHUF
- 4: Sample $\sigma \sim \text{Unif}(\mathcal{S}_N)$, $\pi \sim \text{Unif}(\mathcal{S}_M)$.
- 5: Set $\sigma_k^m(i) := \sigma((i + \frac{N}{M}\pi(m)) \bmod N)$ for all $m \in [M]$, $i \in [N]$.
- 6: **else** ▷ Minibatch RR
- 7: Sample $\sigma_k^m \sim \text{Unif}(\mathcal{S}_N)$ independently and locally, for all $m \in [M]$.
- 8: **end if**
- 9: **for** $i \in [\frac{N}{B}]$ **do**
- 10: Update $\mathbf{x}_{k,i} := \mathbf{x}_{k,i-1} - \underbrace{\frac{\eta}{M} \sum_{m=1}^M \frac{1}{B} \sum_{j=(i-1)B+1}^{iB} \nabla f_{\sigma_k^m(j)}^m(\mathbf{x}_{k,i-1})}_{\text{averaging done locally}}$.
- 11: **end for**
- 12: $\mathbf{x}_{k+1,0} := \mathbf{x}_{k, \frac{N}{B}}$.
- 13: **end for**
- 14: **return** the last iterate $\mathbf{x}_{K, \frac{N}{B}}$.

wise.

Assumption 5.3 (Objective-wise inter-machine deviation). *There exist $\tau \geq 0$ and $\rho \geq 1$ such that*

$$\frac{1}{M} \sum_{m=1}^M \|\nabla F^m(\mathbf{x})\| \leq \tau + \rho \|\nabla F(\mathbf{x})\|, \text{ for all } \mathbf{x} \in \mathbb{R}^d.$$

Assumption 5.3 models the heterogeneity by bounding the mean of $\|\nabla F^m\|$ by a constant plus a multiplicative factor times $\|\nabla F\|$. The assumption includes the homogeneous case (i.e., $F^1 = \dots = F^M = F$) by $\tau = 0$ and $\rho = 1$. Assumption 5.3 is weaker than many other heterogeneity assumptions in the literature (e.g., Karimireddy et al. (2020)); see Appendix B.1 for detailed comparisons.

Assumption 5.3 measures heterogeneity by only considering the local objectives F^m , not the local components f_i^m . We consider a more fine-grained notion of heterogeneity in Assumption 5.4:

Assumption 5.4 (Component-wise inter-machine deviation). *For all $i \in [N]$, let $\bar{f}_i :=$*

$\frac{1}{M} \sum_{m=1}^M f_i^m$. There exist $\lambda \geq 0$ such that for all $m \in [M]$ and $i \in [N]$,

$$\|\nabla f_i^m(\mathbf{x}) - \nabla \bar{f}_i(\mathbf{x})\| \leq \lambda, \text{ for all } \mathbf{x} \in \mathbb{R}^d.$$

Assumption 5.4 states that the gradients of the i -th components of local machines are “close” to each other. The assumption subsumes the component-wise homogeneous setting, i.e., $f_i^1 = f_i^2 = \dots = f_i^M$, by $\lambda = 0$. In distributed learning, this choice corresponds to the setting where each machine has the same training dataset. Assumption 5.4 with $\lambda > 0$ is also relevant to the case where each device has a slightly perturbed (e.g., by data augmentation techniques) version of a certain dataset. It is straightforward to check that Assumption 5.4 implies Assumption 5.3 with $\tau = \lambda$ and $\rho = 1$.

We conclude this section by defining the function classes we study in this paper.

Definition 5.5 (Function classes). *We consider two classes of global objective functions F , also taking into account their local objectives F^m and local components f_i^m . We assume throughout that f_i^m are differentiable and F is bounded from below.*

$$\begin{aligned} \mathcal{F}_{\text{obj}}(L, \mu, \nu, \tau, \rho) &:= \left\{ F \mid F \text{ is } \mu\text{-PL}; f_i^m \text{ are } L\text{-smooth}; \right. \\ &\quad \left. F, F^m, f_i^m \text{ satisfy Assumptions 5.2 \& 5.3} \right\}, \\ \mathcal{F}_{\text{cmp}}(L, \mu, \nu, \lambda) &:= \left\{ F \mid F \text{ is } \mu\text{-PL}; f_i^m \text{ are } L\text{-smooth}; \right. \\ &\quad \left. F, F^m, f_i^m \text{ satisfy Assumptions 5.2 \& 5.4} \right\}. \end{aligned}$$

Notice that $\mathcal{F}_{\text{obj}}(L, \mu, \nu, \tau, \rho) \supset \mathcal{F}_{\text{cmp}}(L, \mu, \nu, \tau)$ for any $\rho \geq 1$. We only make the PL assumption on the global objective F , not on the local objectives F^m nor on the local components f_i^m . Using L and μ , we define the condition number $\kappa := L/\mu \geq 1$.

5.3 Convergence analysis of minibatch and local RR

5.3.1 Upper bound for minibatch RR

We first begin with the convergence result for minibatch RR on $\mathcal{F}_{\text{obj}}(L, \mu, \nu, \tau, \rho)$, which exhibits a faster large-epoch rate compared to the single-machine setting. For upper bounds, we use $\tilde{O}(\cdot)$ to hide universal constants and logarithmic factors of $\frac{1}{\delta}$, M , N , K , and B .

Theorem 5.6 (Upper bound for minibatch RR). *Suppose that minibatch RR has parameters satisfying Assumption 5.1. For any $F \in \mathcal{F}_{\text{obj}}(L, \mu, \nu, \tau, \rho)$, consider running the algorithm using step-size $\eta = \frac{B \log(MNK^2)}{\mu NK}$ for epochs $K \geq 6\kappa \log(MNK^2)$. Then, with probability at least $1 - \delta$,*

$$F(\mathbf{x}_{K, \frac{N}{B}}) - F^* \leq \frac{F(\mathbf{x}_0) - F^*}{MNK^2} + \tilde{O}\left(\frac{L^2}{\mu^3} \frac{\nu^2}{MNK^2}\right). \quad (5.3)$$

Proof. The proof is in Appendix B.4.2. The key challenge in the convergence analysis of our shuffling-based method stems from the indices sampled within an epoch being dependent on each other. For example, if f_1^m is accessed already, then the index $i = 1$ will not be used in later iterations of the epoch; this dependence significantly complicates the analysis. Our approach starts with realizing that for any permutation σ , $\sum_{i=1}^N f_{\sigma(i)}^m = NF^m$. We decompose gradients $\nabla f_{\sigma_k^m(j)}^m(\mathbf{x}_{k,i-1})$ (see Line 10 of Algorithm 5.2) into $\nabla f_{\sigma_k^m(j)}^m(\mathbf{x}_{k,0})$ plus noise, then aggregate all updates over an epoch to get “one big step of GD plus noise”: $\mathbf{x}_{k+1,0} = \mathbf{x}_{k,0} - \eta N \nabla F(\mathbf{x}_{k,0}) + \eta^2 \mathbf{r}_k$. We bound the noise $\mathbf{r}_k$ using Lemma B.1 (Appendix B.4.6), which is our extension of the Hoeffding-Serfling inequality to the mean of M independent without-replacement sums of vectors; the lemma might be of independent interest too. Lemma B.1 shows that averaging accumulated gradients over M machines reduces variance by M , which leads to the reduction by a factor of M in the bound (5.3). $\square$

Theorem 5.6 shows that for large enough epochs $K \gtrsim \kappa$, minibatch RR converges at a rate of $\tilde{O}(\frac{L^2 \nu^2}{\mu^3 MNK^2})$, with high probability. Compared to the large-epoch rate $\tilde{O}(\frac{L^2 \nu^2}{\mu^3 NK^2})$ of single-machine RR (e.g., Ahn et al. (2020)), we see an additional factor M in the denominator, which

highlights the advantage of multiple machines. If we compare against the with-replacement counterpart, it is known that for strongly convex and smooth F , the optimal convergence rate of minibatch SGD is $\Theta(\frac{\nu^2}{\mu MNK})$,⁵ which is worse than our bound (5.3) if $K \gtrsim \kappa^2$. Also notable is that the convergence rate does *not* depend on the heterogeneity constants (i.e., τ and ρ from Assumption 5.3) of the local objective functions. This observation that minibatch RR is “immune” to heterogeneity is consistent with minibatch SGD in the with-replacement setting (Woodworth et al., 2020b).

Epoch vs communication complexity. One might wonder why (5.3) does not have the batch size B . In (5.3), we wrote convergence rates in terms of epochs K , which captures the gradient computation complexity because the same number of gradients are evaluated in a single epoch regardless of B . If we are interested in communication complexity instead, we can write (5.3) in terms of the number of communication rounds $R := \frac{NK}{B}$ and get a rate of $\tilde{O}(\frac{L^2 \nu^2 N}{\mu^3 MB^2 R^2})$. From these, we can also discuss the *overall cost* of the algorithm. If the cost of a communication round is c_c , and the cost of local gradient computations over an epoch is c_e , then the total cost to obtain an ϵ -accurate solution is

$$C_{\text{minibatch}}(\epsilon) = \tilde{O}\left(\frac{c_c \nu \sqrt{N}}{B \sqrt{M} \epsilon} + \frac{c_e \nu}{\sqrt{MN} \epsilon}\right), \quad (5.4)$$

omitting L and μ for simplicity. The total cost shows that there is essentially no harm increasing the batch size B in minibatch RR, as we can get more accurate estimates of true gradients as B becomes larger. In the next subsection, we will see that this is not the case in local RR.

What about $K \lesssim \kappa$? We remark that all upper bounds in this paper hold only for the “large-epoch” regime, where $K \gtrsim \kappa$. Such requirements are common in the literature of without-replacement SGD (Haochen and Sra, 2019b; Nagaraj et al., 2019; Rajput et al., 2020;

⁵The optimal rate for (with-replacement) SGD after R iterations is $\Theta(\frac{\nu^2}{\mu R})$ (see e.g., Rakhlin et al. (2012)). With-replacement minibatching reduces the variance ν^2 to $\frac{\nu^2}{MB}$, and $R = \frac{NK}{B}$. However, achieving the optimal rate for *last iterates* typically requires carefully designed step-size schemes (Jain et al., 2019).

Ahn et al., 2020), and there is a recent result (Safran and Shamir, 2021) suggesting that faster convergence of without-replacement SGD may *not* be possible in the $K \lesssim \kappa$ regime. We defer a more detailed discussion on this regime to Section 5.4, after Theorem 5.8.

5.3.2 Upper bound for local RR

Next, we are interested in how fast local RR can converge, what is the optimal batch size B , and whether local RR can be as fast as minibatch RR.

Theorem 5.7 (Upper bound for local RR). *Suppose that local RR has parameters satisfying Assumption 5.1. For any $F \in \mathcal{F}_{\text{obj}}(L, \mu, \nu, \tau, \rho)$, consider running the algorithm using step-size $\eta = \frac{\log(MNK^2)}{\mu NK}$ for epochs $K \geq 7\rho\kappa \log(MNK^2)$. Then, with probability at least $1 - \delta$,*

$$F(\mathbf{y}_{K, \frac{N}{B}}) - F^* \leq \frac{F(\mathbf{y}_0) - F^*}{MNK^2} + \tilde{\mathcal{O}} \left(\frac{L^2}{\mu^3} \left(\frac{\nu^2}{MNK^2} + \frac{\nu^2 B}{N^2 K^2} + \frac{\tau^2 B^2}{N^2 K^2} \right) \right). \quad (5.5)$$

Proof. The proof is in Appendix B.4.3. We take the same “big GD step plus noise” approach as in Theorem 5.6; however, due to local updates, bounding the noise is much more involved. In the proof, we obtain the epoch update $\mathbf{y}_{k+1,0} = \mathbf{y}_{k,0} - \eta N \nabla F(\mathbf{x}_{k,0}) + \eta^2 \mathbf{r}_{k,1} + \eta^2 \mathbf{r}_{k,2} - \eta^3 \mathbf{r}_{k,3}$, where $\mathbf{r}_{k,1}$ and $\mathbf{r}_{k,3}$ contain errors introduced by local updates. Noise from local updates accumulates over B iterations, which cannot be remedied by averaging over M machines. They result in two additional terms in the rate (5.5), one from intra-machine variance and the other from heterogeneity. $\square$

5.3.2.1 Discussion of Theorem 5.7

Let us compare our high-probability bound (5.5) with existing in-expectation bounds. For strongly convex F , the corresponding *last-iterate* bound of local SGD is $\tilde{\mathcal{O}}(\frac{L\nu^2}{\mu^2 MNK} + \frac{L^2 \nu^2 B}{\mu^3 N^2 K^2} + \frac{L^2 \tau^2 B^2}{\mu^3 N^2 K^2})$ ⁶ (Khaled et al., 2020; Spiridonoff et al., 2020; Qu et al., 2020). Notice that (5.5) is better than this with-replacement bound when $K \gtrsim \kappa$. For *average iterates*, there are known

⁶Due to differences in assumptions, many existing rates cannot be compared directly. These rates are the ones we consider “comparable” to our bound. See Appendix B.1 for more detailed comparisons.

bounds $\tilde{O}(\frac{\nu^2}{\mu MNK} + \frac{L\nu^2 B}{\mu^2 N^2 K^2} + \frac{L\tau^2 B^2}{\mu^2 N^2 K^2})^6$ (Koloskova et al., 2020; Woodworth et al., 2020b) which are smaller than the last-iterate bound by a factor of κ . It is unclear if averaging iterates could improve our rate, because most such analyses exploit Jensen’s inequality, which we cannot use for nonconvex F .

Dependence on τ and ρ . Out of the two heterogeneity constants τ and ρ (Assumption 5.3), ρ does not appear in (5.5), and it only affects the epoch requirement $K \gtrsim \rho\kappa$. Consider the case $\tau = 0$ and $\rho > 1$, which is heterogeneous but in the “interpolation regime,” because $\nabla F^m(\mathbf{x}) = \mathbf{0}$ whenever $\nabla F(\mathbf{x}) = \mathbf{0}$. In such a case, the rate (5.5) is equal to the homogeneous case.

Using $B = \Theta(N)$ is no better than single-machine. A close look at Theorem 5.7 reveals a rather surprising fact. Even in the homogeneous case ($\tau = 0$), if we choose $B = \Theta(N)$, then local RR converges at the rate of $\tilde{O}(\frac{1}{NK^2})$: the same rate as the single-machine RR! In Section 5.4, we show that this observation is not due to a suboptimal analysis; the rate $\tilde{O}(\frac{1}{NK^2})$ is tight for $B = \Theta(N)$.

Trade-off in the choice of B . As done for Theorem 5.6, we can compute from (5.5) that the total cost of local RR for ϵ -accuracy is (omitting L and μ for simplicity)

$$C_{\text{local}}(\epsilon) = \tilde{O}\left(c_c\left(\frac{\nu\sqrt{N}}{B\sqrt{M\epsilon}} + \frac{\nu}{\sqrt{B\epsilon}} + \frac{\tau}{\sqrt{\epsilon}}\right) + c_e\left(\frac{\nu}{\sqrt{MN\epsilon}} + \frac{\nu\sqrt{B}}{N\sqrt{\epsilon}} + \frac{\tau B}{N\sqrt{\epsilon}}\right)\right). \quad (5.6)$$

Note that for local RR, there exists a trade-off between communication and epoch complexity in the choice of B . If B is too small, this reduces the number of epochs required but increases communication costs. On the other hand, if B is too large, this reduces communication rounds but errors that accumulate in local updates get severer, resulting in the need for more epochs. Hence, the optimal choice of B must balance the two complexity measures. The existence of this trade-off is indeed different from minibatch RR where larger B always reduces the total cost $C_{\text{minibatch}}(\epsilon)$.

When can local RR match minibatch RR? Comparing the convergence rates (5.3) and

(5.5), we can identify some regimes in which local RR converges as fast as minibatch RR. In a nutshell, if machines are not too heterogeneous and communication happens frequently, then local RR can have the same upper bound as minibatch RR. For example, if B is chosen to be a constant, $M \lesssim N$, and $\tau \lesssim \nu\sqrt{N/M}$, then the $\tilde{O}(\frac{L^2\nu^2}{\mu^3 MNK^2})$ term in (5.5) becomes the dominating factor and hence matches (5.3). Another example of such a regime is when $B \lesssim \frac{N}{M}$ and $\tau \lesssim \nu\sqrt{M/N}$. Note that this comparison assumes that the same values of B are chosen for both algorithms. Also, such “frequent communication” regimes are favorable if the communication cost c_c is small.

Can local RR ever beat minibatch RR? The upper bounds (5.3) and (5.5) indicate that local RR is always no better than minibatch RR, at least for the function class $\mathcal{F}_{\text{obj}}(L, \mu, \nu, \tau, \rho)$. This is in fact consistent with Woodworth et al. (2020a,b), because the authors identify a regime where local SGD performs better than minibatch SGD for convex objective functions, but fail to do so for strongly convex functions. However, as was also pointed out in Woodworth et al. (2020a), there is a simple extreme scenario in which local RR can be faster: when $\nu \approx \tau \approx 0$ and $\rho \approx 1$. In this case, we have $f_i^m \approx F$ for all m and i , so local RR corresponds to NK steps of GD, whereas minibatch RR corresponds to $\frac{NK}{B}$ steps of GD. Clearly, local RR will converge faster, exploiting the advantage of more updates. Finding out other such regimes is an important future direction.

5.4 Matching lower bounds

In Section 5.3, we presented large-epoch upper bounds (i.e., for $K \gtrsim \kappa$) for constant step-size minibatch and local RR. In this section, we prove matching lower bounds to show that the upper bounds are tight, in all factors except L and μ . We use $\Omega(\cdot)$ to hide universal constants in lower bounds.

5.4.1 Lower bound for minibatch RR

Theorem 5.8 (Lower bound for minibatch RR). *Suppose that minibatch RR has parameters satisfying Assumption 5.1. Additionally, assume that N is a multiple of 2. Then, there exist large enough constants $c_1, c_2 > 0$ such that the following holds: For L and μ satisfying $\kappa = \frac{L}{\mu} \geq c_1$, there exists a function $F \in \mathcal{F}_{\text{cmp}}(L, \mu, \nu, 0)$ such that for any constant step-size η ,*

$$\mathbb{E} \left[F(\mathbf{x}_{K, \frac{N}{K}}) - F^* \right] = \begin{cases} \Omega \left(\frac{\nu^2}{\mu M N K} \right) & \text{if } K < c_2 \kappa, \\ \Omega \left(\frac{\nu^2}{\mu M N K^2} \right) & \text{if } K \geq c_2 \kappa. \end{cases} \quad (5.7)$$

Proof. We prove Theorem 5.8 in Appendix B.5. The proof is an extension of Rajput et al. (2020); Safran and Shamir (2020, 2021) to minibatch RR. We will sketch some key intuitions after Theorem 5.10. $\square$

First notice that the function F is from $\mathcal{F}_{\text{cmp}}(L, \mu, \nu, 0)$, where all the machines are component-wise homogeneous. As seen in Definition 5.5, $\mathcal{F}_{\text{cmp}}(L, \mu, \nu, 0) \subset \mathcal{F}_{\text{obj}}(L, \mu, \nu, \tau, \rho)$ for any $\tau \geq 0$ and $\rho \geq 1$, so Theorem 5.8 provides a lower bound for $\mathcal{F}_{\text{cmp}}(\cdot)$ and $\mathcal{F}_{\text{obj}}(\cdot)$, with arbitrary heterogeneity constants. We assume that N is even because we construct functions g_1 and g_2 such that $f_i^m := g_1$ if $i \leq \frac{N}{2}$, and $f_i^m := g_2$ if $i > \frac{N}{2}$. One can remove this assumption by using a zero function when N is odd (see e.g., Safran and Shamir (2020)). It is rather unsatisfactory that our theorem requires large enough constants c_1 and c_2 ; we believe a tighter analysis can relax this restriction.

Theorem 5.8 proves lower bounds for two different regimes: $K \gtrsim \kappa$ and $K \lesssim \kappa$. In the large-epoch regime ($K \gtrsim \kappa$), we can observe that the lower bound $\Omega(\frac{\nu^2}{\mu M N K^2})$ matches the upper bound (5.3) in Theorem 5.6, modulo a factor of κ^2 . Tightening the κ^2 gap between upper and lower bounds is left for future work. In the small-epoch regime ($K \lesssim \kappa$), we observe that the lower bound $\Omega(\frac{\nu^2}{\mu M N K})$ exactly matches the convergence rate of (with-replacement) minibatch SGD; hence, the lower bound implies that minibatch RR has no hope for faster convergence than minibatch SGD, at least in the constant step-size and small-epoch regime. This observation is in line with Safran and Shamir (2021).

Upper bounds for $K \lesssim \kappa$? Even for single-machine RR ($M = 1$), proving an upper bound that matches the small-epoch lower bound $\Omega(\frac{\nu^2}{\mu NK})$ still remains a challenge. Nagaraj et al. (2019, Theorem 2) prove an upper bound for non-quadratic strongly convex functions that matches $\Omega(\frac{\nu^2}{\mu NK})$ if $NK \gtrsim \kappa^2$; however, they use suffix averaging, so it is not directly comparable to Theorem 5.8 which considers last iterates. Safran and Shamir (2021) prove upper bounds for quadratic strongly convex functions, but assume that their Hessian matrices commute. For noncommutative cases, proving a small-epoch upper bound seems to require some form of matrix AM-GM inequalities, whose availability is an open problem (Recht and Ré, 2012b; Lai and Lim, 2020; De Sa, 2020; Yun et al., 2021).

Remark 5.9 (Strong convexity in construction). *We note that all lower bounds in this paper are constructed with strongly convex functions, a stronger assumption than PL functions (5.2). Thus, our lower bounds are also applicable to strong convexity counterparts of $\mathcal{F}_{\text{obj}}(\cdot)$ and $\mathcal{F}_{\text{cmp}}(\cdot)$.*

5.4.2 Lower bounds for local RR

In this subsection, we present lower bounds for local RR. We prove two bounds that correspond to homogeneous and heterogeneous cases. By combining the two bounds, we get a lower bound that matches our upper bound (5.5) in Theorem 5.7 up to a factor of κ^2 .

Theorem 5.10 (Lower bound for local RR: homogeneous case). *Suppose that local RR has parameters satisfying Assumption 5.1. Additionally, assume that B is a multiple of 4. Then, there exist large enough constants $c_3, c_4 > 0$ such that the following holds: For L and μ satisfying $\kappa = \frac{L}{\mu} \geq c_3$, there exists a function $F \in \mathcal{F}_{\text{cmp}}(L, \mu, \nu, 0)$ such that for any constant step-size η ,*

$$\mathbb{E} \left[F(\mathbf{y}_{K, \frac{N}{K}}) - F^* \right] = \begin{cases} \Omega \left(\frac{\nu^2}{\mu MNK} \right) & \text{if } K < \max \left\{ c_4 \kappa, \frac{MB}{N} \right\}, \\ \Omega \left(\frac{\nu^2}{\mu MNK^2} + \frac{\nu^2 B}{\mu N^2 K^2} \right) & \text{if } K \geq \max \left\{ c_4 \kappa, \frac{MB}{N} \right\}. \end{cases} \quad (5.8)$$

Proof. The proof is in Appendix B.7. For the large-epoch lower bounds in Theorems 5.8 and 5.10, we use “skewed” quadratics $f_i^m(x) = (L1_{x \leq 0} + \mu 1_{x > 0}) \frac{x^2}{2} + z_i \nu x$, where $z_i = +1$ if

$i \leq \frac{N}{2}$ and $z_i = -1$ otherwise. For $x \approx 0$, the imbalance results in a “drift” towards positive x , whose strength is approximately proportional to the absolute value of partial sums of random permutations over $\frac{N}{2} + 1$ ’s and $\frac{N}{2} - 1$ ’s. By averaging the sums over M machines (minibatch RR), their absolute values shrink by $\frac{1}{\sqrt{M}}$; in contrast, if each machine makes local updates (local RR), the magnitude of the drift cannot be reduced with M , because we average after local iterates already have taken B “big” steps. The proof uses techniques from Rajput et al. (2020). $\square$

Proposition 5.11 (Lower bound for local RR: heterogeneous case). *Suppose that local RR has parameters satisfying Assumption 5.1. Additionally, assume that B is a multiple of 2 and $\kappa = \frac{L}{\mu} \geq 2$. Then, there exists a function $F \in \mathcal{F}_{\text{obj}}(L, \mu, 0, \tau, 1)$ such that for any constant step-size η ,*

$$\mathbb{E} \left[F(\mathbf{y}_{K, \frac{N}{K}}) - F^* \right] = \Omega \left(\frac{\tau^2 B^2}{\mu N^2 K^2} \right). \quad (5.9)$$

Proof. We note that Proposition 5.11 is almost identical to Theorem II of Karimireddy et al. (2020); however, we provide a proof specific to our algorithm in Appendix B.9. $\square$

Theorem 5.10 constructs a component-wise homogeneous function from $\mathcal{F}_{\text{cmp}}(L, \mu, \nu, 0)$ and Proposition 5.11 constructs a heterogeneous function from $\mathcal{F}_{\text{obj}}(L, \mu, 0, \tau, 1)$. Since $\mathcal{F}_{\text{cmp}}(L, \mu, \nu, 0) \cup \mathcal{F}_{\text{obj}}(L, \mu, 0, \tau, 1) \subset \mathcal{F}_{\text{obj}}(L, \mu, \nu, \tau, \rho)$ for any $\rho \geq 1$, combining (5.8) and (5.9) for the $K \geq \max\{c_4 \kappa, \frac{MB}{N}\}$ case gives a lower bound

$$\Omega \left(\max \left\{ \frac{\nu^2}{\mu M N K^2} + \frac{\nu^2 B}{\mu N^2 K^2}, \frac{\tau^2 B^2}{\mu N^2 K^2} \right\} \right),$$

that matches the large-epoch upper bound (5.5) in Theorem 5.7, up to a factor of κ^2 . When $\kappa N \gtrsim MB$, $c_4 \kappa$ becomes the dominating term in the max, in which case the threshold in (5.8) is $\Theta(\kappa)$. Tightening the κ^2 gap as well as removing additional requirements such as $\kappa \geq c_3$ and $\kappa N \gtrsim MB$ are left for future work.

Using $B = \Theta(N)$ does not help, indeed. In Section 5.3.2.1, we observed that if $B = \Theta(N)$, then even in the homogeneous case ($\tau = 0$), local RR converges at the rate of $\tilde{O}(\frac{1}{NK^2})$. This

is the same rate as single-machine RR, meaning the efforts by $M - 1$ machines become meaningless. Our lower bound (5.8) shows that $\tilde{O}(\frac{1}{NK^2})$ is in fact the best we can hope for (treating L and μ as constants). In order to make the best use of M machines, B should be smaller than $\Theta(N)$, as suggested in Section 5.3.2.1. In an existing work, Mishchenko et al. (2021) consider local RR with $B = N$ as a special case of a proximal algorithm. In Theorem 8 of Mishchenko et al. (2021), the authors claim “the convergence bound improves with the number of devices involved” because the bound has a factor of M in the denominator. However, at least under our assumption, this is not the case; if we apply our Assumption 5.3 to upper-bound their σ_* , the term “ $N\sigma_*^2$ ” in the numerator grows linearly with M . Hence, our bounds do not contradict Mishchenko et al. (2021); see Appendix B.1 for details.

Remark 5.12 (Small-epoch bound is likely loose). *We note that while we focused on deriving a matching large-epoch lower bound, we did not try hard to tighten the small-epoch lower bound. Our small-epoch lower bound in (5.8) misses a term (such as $\frac{\nu^2 B}{\mu N^2 K^2}$) that corresponds to the error from local updates. We leave investigations on small-epoch lower and upper bounds for future work.*

5.5 Synchronized shuffling: how to bypass lower bounds

Recall from the total complexity of minibatch RR (5.4) that the total cost shrinks with a factor of $\frac{1}{\sqrt{M}}$. Using M machines, we are only getting a $\sqrt{M}$ -factor speedup. Ideally, we hope to see a *linear speedup*, i.e., cost inverse proportional to M . Hence, Theorem 5.6 falls short of achieving this goal, and our lower bound in Theorem 5.8 confirms that linear speedup is indeed impossible.

In this section, we show that the desired linear speedup is possible, at least in some special cases. We consider the component-wise near-homogeneous case (i.e., Assumption 5.4 with small λ) and discuss how a simple modification to minibatch and local RR can let us “break” the lower bounds and achieve linear speedup. This comes at a cost of broadcasting permutations: at the beginning of the k -th epoch, the server samples $\sigma \sim \text{Unif}(\mathcal{S}_N)$ and

$\pi \sim \text{Unif}(\mathcal{S}_M)$, and broadcasts them to the machines. Then, local machines choose their permutations σ_k^m to be *shifted* versions of σ ,⁷ i.e., $\sigma_k^m(i) := \sigma\left(\left(i + \frac{N}{M}\pi(m)\right) \bmod N\right)$. We call this trick **synchronized shuffling**, denoted as **SyncSHUF**. Please revisit Algorithms 5.1 and 5.2 for the precise descriptions of the modified algorithms **local RR with SyncSHUF** and **minibatch RR with SyncSHUF**, respectively.

The intuition why this should help is simple. In the proof of RR, we aggregate the component gradients over an epoch (i.e., N iterations) to write it as a full gradient plus noise. If we are in the component-wise homogeneous setting and permutations are synchronized, then instead of aggregating N component gradients on a single machine, we can aggregate $\frac{N}{M}$ component gradients on M machines to get a full gradient. This allows us to reduce the “noise” from without-replacement sampling. We emphasize here that we do not necessarily set $B = \frac{N}{M}$ to get a full gradient every time; our analysis works for arbitrary B and M , as long as both divide N . See Appendix B.2 for a detailed illustration of SyncSHUF; also, see Appendix B.3 for experiments showing its effectiveness.

The idea of synchronized shuffling is similar to approaches in distributed learning that shuffle and partition datasets and distribute them to local machines (see e.g., Lee et al. (2017); Meng et al. (2017)). In contrast, we do not communicate data, but communicate how to permute datasets stored in local machines. Meng et al. (2017, Theorem 3.3) provide an analysis for a distributed method similar to minibatch RR, but fail to show convergence to global minima in strongly convex cases. We also note that an independent concurrent result (Szlendak et al., 2021) uses the same idea as SyncSHUF to build compressors for communication-efficient distributed optimization.

5.5.1 Upper bounds for minibatch and local RR with SyncSHUF

With SyncSHUF, we can show that the M ’s appearing in the convergence rates ((5.3) and (5.5)) in Theorems 5.6 and 5.7 can be replaced with M^2 , for a more stringent function class

⁷We assume for simplicity that M divides N .

$\mathcal{F}_{\text{cmp}}(\cdot)$ that requires bounded component-wise inter-machine deviation (Assumption 5.4).

Theorem 5.13 (Upper bound for minibatch RR with SYNC SHUF). *Suppose that minibatch RR with SYNC SHUF has parameters satisfying Assumption 5.1. Additionally assume that M divides N . For any $F \in \mathcal{F}_{\text{cmp}}(L, \mu, \nu, \lambda)$, consider running the algorithm using step-size $\eta = \frac{B \log(M^2 N K^2)}{\mu N K}$ for epochs $K \geq 6\kappa \log(M^2 N K^2)$. Then, with probability at least $1 - \delta$,*

$$F(\mathbf{x}_{K, \frac{N}{B}}) - F^* \leq \frac{F(\mathbf{x}_0) - F^*}{M^2 N K^2} + \tilde{\mathcal{O}} \left(\frac{L^2}{\mu^3} \left(\frac{\nu^2}{M^2 N K^2} + \frac{\lambda^2}{M K^2} \right) \right). \quad (5.10)$$

The proof of Theorem 5.13 is presented in Appendix B.4.4. One can check that if the component-wise deviation constant λ satisfies $\lambda \lesssim \frac{\nu}{\sqrt{MN}}$ (i.e., near-homogeneous), then the rate (5.10) becomes $\tilde{\mathcal{O}}(\frac{1}{M^2 N K^2})$. It is then easy to confirm that M machines reduce total costs by $\frac{1}{M}$ —a linear speedup.

A similar speedup can be shown for local RR. In Appendix B.4.5, we prove that

Theorem 5.14 (Upper bound for local RR with SYNC SHUF). *Suppose that local RR with SYNC SHUF has parameters satisfying Assumption 5.1. Additionally assume that M divides N . For any $F \in \mathcal{F}_{\text{cmp}}(L, \mu, \nu, \lambda)$, consider running the algorithm with step-size $\eta = \frac{\log(M^2 N K^2)}{\mu N K}$ for epochs $K \geq 7\kappa \log(M^2 N K^2)$. Then, with probability at least $1 - \delta$,*

$$F(\mathbf{y}_{K, \frac{N}{B}}) - F^* \leq \frac{F(\mathbf{y}_0) - F^*}{M^2 N K^2} + \tilde{\mathcal{O}} \left(\frac{L^2}{\mu^3} \left(\frac{\nu^2}{M^2 N K^2} + \frac{\nu^2 B}{N^2 K^2} + \frac{\lambda^2 B^2}{N^2 K^2} + \frac{\lambda^2}{M K^2} \right) \right). \quad (5.11)$$

We can similarly check that if $B \lesssim \frac{N}{M^2}$ and $\lambda \lesssim \frac{\nu}{\sqrt{MN}}$, i.e., frequent communication and near-homogeneity, then the $\tilde{\mathcal{O}}(\frac{1}{M^2 N K^2})$ term dominates in (5.11), and hence gives a linear speedup that matches the best rate of minibatch RR with SYNC SHUF (5.10). Nevertheless, we note again that for local RR, such a small B is favorable only when the communication cost c_c is small (recall (5.6)).

5.6 Conclusion

We studied convergence bounds for local RR and minibatch RR, which are the practical without-replacement versions of local and minibatch SGD studied in the theory literature. For smooth functions satisfying the Polyak-Łojasiewicz condition, we showed large-epoch convergence bounds for minibatch and local RR that are faster than their with-replacement counterparts. We also proved matching lower bounds showing that our convergence analysis is tight. We also proposed a simple modification called synchronized shuffling that leads to convergence rates faster than our lower bounds in near-homogeneous settings. Immediate future research directions include extension to small-epoch regimes, as well as to general convex and nonconvex functions.

Ethics Statement

This paper develops theoretical guarantees for popular distributed stochastic optimization algorithms. Therefore, the authors do not see any particular concerns related to its ethical aspects or future societal consequences.

Reproducibility Statement

This paper is a theoretical work, without any experimental results. Definitions and assumptions are provided in Section 5.2. Our theoretical contributions as well as some additionally required assumptions are clearly stated in Sections 5.3, 5.4, and 5.5. Complete proofs of all the theorems are provided in the appendix.

6 RANDOM RESHUFFLING: IS RANDOM OPTIMAL?

A recent line of ground-breaking results for Random Reshuffling has corroborated a widely observed phenomenon: using random permutations of data every epoch offers faster convergence than vanilla SGD, which does sampling with-replacement. However, is random optimal? We show that this depends heavily on what functions we are optimizing, and the convergence gap between optimal and random permutations can vary from exponential to nonexistent. We first show that for 1-dimensional strongly convex functions, with smooth second derivatives, there exist permutations that offer exponentially faster convergence compared to random. However, for general strongly convex functions, random permutations are optimal. Finally, we show that for quadratic, strongly-convex functions, there are easy-to-construct permutations that lead to accelerated convergence compared to random. Our results suggest that a general convergence characterization of optimal permutations cannot capture the nuances of individual function classes, and can mistakenly indicate that one cannot do much better than random.

6.1 Introduction

Finite sum optimization seeks to solve the following:

$$\min_{\mathbf{x}} F(\mathbf{x}) := \frac{1}{n} \sum_{i=1}^n f_i(\mathbf{x}). \quad (6.1)$$

Stochastic gradient descent (SGD) approximately solves finite sum problems, by iteratively updating the optimization variables according to the following rule:

$$\mathbf{x}_{t+1} := \mathbf{x}_t - \alpha \nabla f_{\sigma_t}(\mathbf{x}_t), \quad (6.2)$$

This chapter presents our paper "Permutation-Based SGD: Is Random Optimal?" (Rajput et al., 2022).

where α is the step size and $\sigma_t \in [n] = \{1, 2, \dots, n\}$ is the index of the function sampled at iteration t . There exist various ways of sampling σ_t , with the most common being with- and without-replacement sampling. In the former, σ_t is uniformly chosen at random from $[n]$, and for the latter, σ_t represents the t -th element of a random permutation of $[n]$. We henceforth refer to these two SGD variants as vanilla and shuffling-based, respectively.

Although shuffling-based SGD has been widely observed to perform better in practice (Bottou, 2009; Recht and Ré, 2012a; Recht and Re, 2013), the vanilla version has attracted the vast majority of theoretical analysis. This is because of the fact that at each iteration, in expectation the update is a scaled version of the true gradient, allowing for simple performance analyses of the algorithm, *e.g.*, see (Bubeck et al., 2015).

Shuffling-based SGD has resisted a tight analysis for a long time. However, a recent line of breakthrough results provides the first tight convergence guarantees for several classes of convex functions F (Nagaraj et al., 2019; Safran and Shamir, 2020; Rajput et al., 2020; Mishchenko et al., 2020; Ahn et al., 2020; Nguyen et al., 2020). These recent studies mainly focus on two variants of shuffling-based SGD where (1) a new random permutation is sampled at each epoch (also known as RANDOM RESHUFFLE) (Nagaraj et al., 2019; Safran and Shamir, 2020; Rajput et al., 2020), and (2) a random permutation is sampled once and is reused throughout all SGD epochs (SINGLE SHUFFLE) (Safran and Shamir, 2020; Mishchenko et al., 2020; Ahn et al., 2020).

Perhaps interestingly, RANDOM RESHUFFLE and SINGLE SHUFFLE exhibit different convergence rates and a performance gap that varies across different function classes. In particular, when run for K epochs, the convergence rate for strongly convex functions is $\tilde{O}(1/nK^2)$ for both RANDOM RESHUFFLE and SINGLE SHUFFLE (Nagaraj et al., 2019; Ahn et al., 2020; Mishchenko et al., 2020). However, when run specifically on strongly convex quadratics, RANDOM RESHUFFLE experiences an acceleration of rates, whereas SINGLE SHUFFLE does not (Safran and Shamir, 2020; Rajput et al., 2020; Ahn et al., 2020; Mishchenko et al., 2020). All the above rates have been coupled by matching lower bounds, at least up to constants and

	Plain	with FLIPFLOP	
RR	$\Omega\left(\frac{1}{n^2 K^2} + \frac{1}{n K^3}\right)$	$\tilde{O}\left(\frac{1}{n^2 K^2} + \frac{1}{n K^5}\right)$	Thm. 6.10
SS	$\Omega\left(\frac{1}{n K^2}\right)$	$\tilde{O}\left(\frac{1}{n^2 K^2} + \frac{1}{n K^4}\right)$	Thm. 6.9
IGD	$\Omega\left(\frac{1}{K^2}\right)$	$\tilde{O}\left(\frac{1}{n^2 K^2} + \frac{1}{K^3}\right)$	Thm. 6.11

Table 6.1: Convergence rates of RANDOM RESHUFFLE (RR), SINGLE SHUFFLE (SS) and INCREMENTAL GRADIENT DESCENT (IGD) on strongly convex quadratics: Plain vs. FLIPFLOP. Lower bounds for the “plain” versions are taken from (Safran and Shamir, 2020). When $n \gg K$, that is when the training set is much larger than the number of epochs, which arguably is the case in practice, the convergence rates of RANDOM RESHUFFLE, SINGLE SHUFFLE, and INCREMENTAL GRADIENT DESCENT are $\Omega(\frac{1}{n K^3})$, $\Omega(\frac{1}{n K^2})$, and $\Omega(\frac{1}{K^2})$ respectively. On the other hand, by combining these methods with FLIPFLOP the convergence rates become faster, *i.e.*, $\tilde{O}(\frac{1}{n K^5})$, $\tilde{O}(\frac{1}{n K^4})$, and $\tilde{O}(\frac{1}{K^3})$, respectively.

sometimes log factors (Safran and Shamir, 2020; Rajput et al., 2020).

Algorithm 6.1 Shuffling-based SGD variants

Input: Initialization $\mathbf{x}_0^1$, step size α , epochs K

```

1:  $\sigma =$  a random permutation of  $[n]$ 
2: for  $k = 1, \dots, K$  do
3:   if IGD then
4:      $\sigma^k = (1, 2, \dots, n)$ 
5:   else if SINGLE SHUFFLE then
6:      $\sigma^k = \sigma$ 
7:   else if RANDOM RESHUFFLE then
8:      $\sigma^k =$  a random permutation of  $[n]$ 
9:   end if

10:  if FLIPFLOP and  $k$  is even then
11:     $\sigma^k =$  reverse of  $\sigma^{k-1}$ 
12:  end if

13:  for  $i = 1, \dots, n$  do
14:     $\mathbf{x}_i^k := \mathbf{x}_{i-1}^k - \alpha \nabla f_{\sigma^k}(\mathbf{x}_{i-1}^k)$ 
15:  end for
16:   $\mathbf{x}_0^{k+1} := \mathbf{x}_n^k$ 
17: end for
```

}

Epoch k

From the above we observe that reshuffling at the beginning of every epoch may not always help. But then there are cases where RANDOM RESHUFFLE is faster than SINGLE SHUFFLE, implying that certain ways of generating permutations are more suited for certain

subfamilies of functions.

The goal of our paper is to take a first step into exploring the relationship between convergence rates and the particular choice of permutations. We are particularly interested in understanding if random permutations are as good as optimal, or if SGD can experience faster rates with carefully crafted permutations. As we see in the following, the answer the above is not straightforward, and depends heavily on the function class at hand.

Our Contributions: For the purpose of this chapter, we override the definition of *shuffling-based SGD* and define it to be any variant of the iterates in (6.2), where a permutation of the n functions, at the start of each epoch, can be generated deterministically, randomly, or with a combination of the two. For example, SINGLE SHUFFLE, RANDOM RESHUFFLE, and incremental gradient descent (IGD), are all shuffling-based SGD variants (see Algorithm 6.1).

We first want to understand—even in the absence of computational constraints in picking the optimal permutations—what is the fastest rate one can get for shuffling-based SGD? In other words, are there permutations that are better than random in the eyes of SGD?

Perhaps surprisingly, we show that there exist permutations that may offer up to exponentially faster convergence than random permutations, but for a limited set of functions. Specifically, we show this for 1-dimensional functions (Theorem 6.5). However, such exponential improvement is no longer possible in higher dimensions (Theorem 6.6), or for general strongly convex objectives (Theorem 6.7), where random is optimal. The above highlight that an analysis of how permutations affect convergence rates needs to be nuanced enough to account for the structure of functions at hand. Otherwise, in lieu of further assumptions, random permutations may just appear to be as good as optimal.

In this work, we further identify a subfamily of convex functions, where there exist easy-to-generate permutations that lead accelerated convergence. We specifically introduce a new technique, FLIPFLOP, which can be used in conjunction with existing shuffling-based

methods, *e.g.*, RANDOM RESHUFFLE, SINGLE SHUFFLE, or INCREMENTAL GRADIENT DESCENT, to provably improve their convergence on quadratic functions (Theorems 6.9, 6.10, and 6.11). The way that FLIPFLOP works is rather simple: every even epoch uses the *flipped* (or reversed) version of the previous epoch’s permutation. The intuition behind why FLIPFLOP leads to faster convergence is as follows. Towards the end of an epoch, the contribution of earlier gradients gets attenuated. To counter this, we flip the permutation for the next epoch so that every function’s contribution is diluted (approximately) equally over the course of two consecutive epochs. FLIPFLOP demonstrates that finding better permutations for specific classes of functions might be computationally easy. We summarize FLIPFLOP’s convergence rates in Table 6.1 and report the results of numerical verification in Section 6.6.2.

Note that in this work, we focus on the dependence of the error on the number of iterations, and in particular, the number of epochs. However, we acknowledge that its dependence on other parameters like the condition number is also very important. We leave such analysis for future work.

Notation: We use lowercase for scalars (a), lower boldface for vectors ($\mathbf{a}$), and upper boldface for matrices ($\mathbf{A}$).

6.2 Related Work

Gurbuzbalaban et al. (2019a); Gürbüzbalaban et al. (2015) provided the first theoretical results establishing that RANDOM RESHUFFLE and INCREMENTAL GRADIENT DESCENT (and hence SINGLE SHUFFLE) were indeed faster than vanilla SGD, as they offered an asymptotic rate of $O(1/K^2)$ for strongly convex functions, which beats the convergence rate of $O(1/nK)$ for vanilla SGD when $K = \Omega(n)$. Shamir (2016b) used techniques from online learning and transductive learning theory to prove an optimal convergence rate of $\tilde{O}(1/n)$ for the first epoch of RANDOM RESHUFFLE (and hence SINGLE SHUFFLE). Later, Haochen and Sra (2019a)

also established a non-asymptotic convergence rate of $\tilde{O}\left(\frac{1}{n^2 K^2} + \frac{1}{K^3}\right)$, when the objective function is quadratic, or has smooth Hessian.

Nagaraj et al. (2019) used a very interesting iterate coupling based approach to give a new upper bound on the error rate of RANDOM RESHUFFLE, thus proving for the first time that for general strongly convex smooth functions, it converges faster than vanilla SGD in all regimes of n and K . This was followed by (Safran and Shamir, 2020), where the authors were able to establish the first lower bounds, in terms of both n and K , for RANDOM RESHUFFLE. However, there was a gap in these upper and lower bounds. The gap in the convergence rates was closed by Rajput et al. (2020), who showed that the upper bound given by Nagaraj et al. (2019) and the lower bound given by Safran and Shamir (2020) were both tight up to logarithmic terms.

For SINGLE SHUFFLE, Mishchenko et al. (2020) and Ahn et al. (2020) showed an upper bound of $\tilde{O}\left(\frac{1}{n K^2}\right)$, which matched the lower bound given earlier by (Safran and Shamir, 2020), up to logarithmic terms. Ahn et al. (2020) and Mishchenko et al. (2020) also proved tight upper bounds for RANDOM RESHUFFLE, with a simpler analysis and using more relaxed assumptions than (Nagaraj et al., 2019) and (Rajput et al., 2020). In particular, the results by Ahn et al. (2020) work under the PL condition and do not require individual component convexity.

INCREMENTAL GRADIENT DESCENT on strongly convex functions has also been studied well in literature (Nedić and Bertsekas, 2001; Bertsekas, 2011; Gurbuzbalaban et al., 2019a). More recently, Nguyen et al. (2020) provide a unified analysis for all shuffling-based algorithms. The dependence of their convergence rates on the number of epochs K is also optimal for INCREMENTAL GRADIENT DESCENT, SINGLE SHUFFLE and RANDOM RESHUFFLE.

There has also been some recent work on the analysis of RANDOM RESHUFFLE on non-strongly convex functions and non-convex functions. Specifically, Nguyen et al. (2020) and Mishchenko et al. (2020) show that even there, RANDOM RESHUFFLE outperforms SGD under certain conditions. Mishchenko et al. (2020) show that RANDOM RESHUFFLE and SINGLE

SHUFFLE beat vanilla SGD on non-strongly convex functions after $\Omega(n)$ epochs, and that RANDOM RESHUFFLE is faster than vanilla SGD on non-convex objectives if the desired error is $O(1/\sqrt{n})$.

Speeding up convergence by combining without replacement sampling with other techniques like variance reduction (Shamir, 2016b; Ying et al., 2020) and momentum (Tran et al., 2021) has also received some attention. In this work, we solely focus on the power of “good permutations” to achieve fast convergence.

6.3 Preliminaries

We will use combinations of the following assumptions:

Assumption 6.1 (Component convexity). $f_i(\mathbf{x})$'s are convex.

Assumption 6.2 (Component smoothness). $f_i(\mathbf{x})$'s are L -smooth, i.e.,

$$\forall \mathbf{x}, \mathbf{y} : \|\nabla f_i(\mathbf{x}) - \nabla f_i(\mathbf{y})\| \leq L\|\mathbf{x} - \mathbf{y}\|.$$

Note that Assumption 6.2 immediately implies that F also has L -Lipschitz gradients:

$$\forall \mathbf{x}, \mathbf{y} : \|\nabla F(\mathbf{x}) - \nabla F(\mathbf{y})\| \leq L\|\mathbf{x} - \mathbf{y}\|.$$

Assumption 6.3 (Objective strong convexity). F is μ -strongly convex, i.e.,

$$\forall \mathbf{x}, \mathbf{y} : F(\mathbf{y}) \geq F(\mathbf{x}) + \langle \nabla F(\mathbf{x}), \mathbf{y} - \mathbf{x} \rangle + \frac{1}{2}\mu\|\mathbf{y} - \mathbf{x}\|^2.$$

Note that Assumption 6.3 implies

$$\forall \mathbf{x}, \mathbf{y} : \langle \nabla F(\mathbf{x}) - \nabla F(\mathbf{y}), \mathbf{x} - \mathbf{y} \rangle \geq \mu\|\mathbf{y} - \mathbf{x}\|^2. \quad (6.3)$$

We denote the condition number by κ , which is defined as $\kappa = \frac{L}{\mu}$. It can be seen easily that $\kappa \geq 1$ always. Let $\mathbf{x}^*$ denote the minimizer of Eq. (6.1), that is, $\mathbf{x}^* = \arg \min_{\mathbf{x}} F(\mathbf{x})$.

We will study shuffling-based algorithms in the *constant* step size regime, that is, the step size is chosen at the beginning of the algorithm, and then remains fixed throughout. We denote the iterate after the i -th iteration of the k -th epoch by $\mathbf{x}_i^k$. Hence, the initialization point is $\mathbf{x}_0^1$. Similarly, the permutation of $(1, \dots, n)$ used in the k -th epoch is denoted by σ^k , and its i -th ordered element is denoted by σ_i^k . Note that if the ambient space is 1-dimensional, then we represent the iterates and the minimizer using non-bold characters, i.e. x_i^k and x^* , to remain consistent with the notation.

In the following, due to lack of space, we only provide sketches of the full proofs, when possible. The full proofs of the lemmas and theorems are provided in the Appendix.

6.4 Exponential Convergence in 1-Dimension

In this section, we show that there exist permutations for Hessian-smooth 1 - dimensional functions that lead to exponentially faster convergence compared to random.

Assumption 6.4 (Component Hessian-smoothness). $f_i(x)$'s have L_H -smooth second derivatives, that is,

$$\forall x, y : |\nabla^2 f_i(x) - \nabla^2 f_i(y)| \leq L_H |x - y|.$$

We also define some instance dependent constants: $G^* := \max_i \|\nabla f_i(\mathbf{x}^*)\|$, $D = \max \left\{ \|\mathbf{x}_0^1 - \mathbf{x}^*\|, \frac{G^*}{2L} \right\}$, and $G = G^* + 2DL$.

Theorem 6.5. *Let Assumptions 6.1, 6.2, 6.3 and 6.4 hold. Let D and G be as defined above. If $\alpha = \frac{\mu}{8n(L^2 + L_H G)}$, then there exists a sequence of permutations $\sigma^1, \sigma^2, \dots, \sigma^K$ such that using those permutations from any initialization point $\mathbf{x}_0^1$ gives the error*

$$|x_n^K - x^*| \leq (D + 4n\alpha G)e^{-CK},$$

where $C = \frac{\mu^2}{16(L^2 + L_H G)}$.

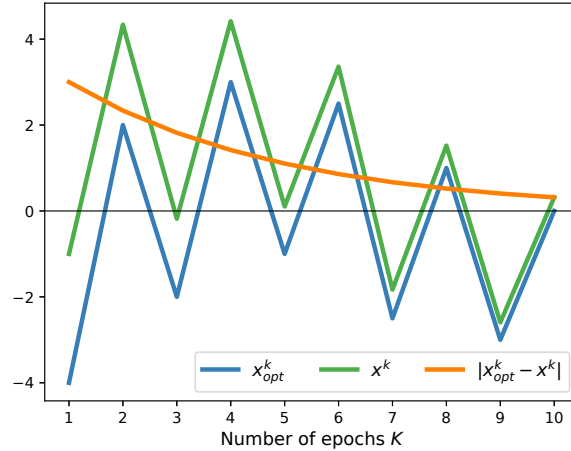


Figure 6.1: (A graphical depiction of Theorem 1's proof sketch.) Assume that the minimizer is at the origin. The proof of Theorem 6.5 first shows that there exists an initialization and a sequence of permutations, such that using those, we get to the exact minimizer. Let the sequence of iterates for this run be x_{opt}^k . Consider a parallel run, which uses the same sequence of permutations, but an arbitrary initialization point. Let this sequence be x_k . The figure shows how x_{opt}^k converges to the exact optima, and the distance between x_{opt}^k and x_k decreases exponentially, leading to an exponential convergence for x_k .

An important thing to notice in the theorem statement is that the sequence of permutations $\sigma^1, \sigma^2, \dots, \sigma^K$ only depends on the function, and not on the initialization point x_0^1 . This implies that for any such function, there exists a sequence of permutations $\sigma^1, \sigma^2, \dots, \sigma^K$, which gives exponentially fast convergence, unconditionally of the initialization. Note that the convergence rate is slower than Gradient Descent, for which the constant 'C' would be larger. However, here we are purely interested in the convergence rates of the best permutations and their (asymptotic) dependence on K .

Proof sketch The core idea is to establish that there exists an initialization point x_0^1 (close to the minimizer x^*), and a sequence of permutations such that that starting from x_0^1 and using that sequence of permutation leads us exactly to the minimizer. Once we have proved this, we show that if two parallel runs of the optimization process are initialized from two different iterates, and they are coupled so that they use the exact same permutations, then they approach each other at an exponential rate. Thus, if we use the same permutation from any initialization point, it will converge to the minimizer at an exponential rate. See

Figure 6.1 for a graphical depiction of this sketch. We note that the figure is not the result of an actual run, but only serves to explain the proof sketch.

6.5 Lower Bounds for Shuffling-based SGD

The result in the previous section leads us to wonder if exponentially fast convergence can be also achieved in higher dimensions. Unfortunately, for higher dimensions, there exist strongly convex quadratic functions for which there does not exist any sequence of permutations that lead to an exponential convergence rate. This is formalized in the following theorem.

Theorem 6.6. *For any $n \geq 4$ (n must be even), there exists a $2n + 1$ -dimensional strongly convex function F which is the mean of n convex quadratic functions, such that for every shuffling-based algorithm with any step size,*

$$\|\mathbf{x}_n^{K-1} - \mathbf{x}^*\|^2 = \Omega\left(\frac{1}{n^3 K^2}\right).$$

This theorem shows that we cannot hope to develop constant step size algorithms that exhibit exponentially fast convergence in multiple dimensions.

Proof sketch In this sketch, we explain a simpler version of the theorem, which works for $n = 2$ and 2-Dimensions. Consider $F(x, y) = \frac{1}{2}f_1(x, y) + \frac{1}{2}f_2(x, y)$ such that

$$f_1(x, y) = \frac{x^2}{2} - x + y, \text{ and } f_2(x, y) = \frac{y^2}{2} - y + x.$$

Hence, $F = \frac{1}{4}(x^2 + y^2)$, and has minimizer at the origin. Each epoch has two possible permutations, $\sigma = (1, 2)$ or $\sigma = (2, 1)$. Working out the details manually, it can be seen that regardless of the permutation, $x_0^{k+1} + y_0^{k+1} > x_0^k + y_0^k$, that is, the sum of the co-ordinates keeps increasing. This can be used to get a bound on the error term $\|[\mathbf{x}^K \quad \mathbf{y}^K]^\top\|^2$.

Next, we show that even in 1-Dimension, individual function convexity might be necessary to obtain faster rates than RANDOM RESHUFFLE.

Theorem 6.7. *There exists a 1-Dimensional strongly convex function F which is the mean of two quadratic functions f_1 and f_2 , such that one of the functions is non-convex. Then, every shuffling-based algorithm with constant step size $\alpha \leq \frac{1}{L}$ gives an error of at least*

$$\|\mathbf{x}_n^{K-1} - \mathbf{x}^*\|^2 = \Omega\left(\frac{1}{K^2}\right).$$

Proof sketch The idea behind the sketch is to have one of the two component functions as strongly concave. This gives it the advantage that the farther away from its maximum the iterate is, the more it pushes the iterate away. Hence, it essentially results in increasing the deviation in each epoch. This leads to a slow convergence rate.

In the setting where the individual f_i may be non-convex, Nguyen et al. (2020) and Ahn et al. (2020) show that SINGLE SHUFFLE, RANDOM RESHUFFLE, and INCREMENTAL GRADIENT DESCENT achieve the error rate of $\tilde{O}(\frac{1}{K^2})$, for a fixed n . In particular, their results only need that the component functions be smooth and hence their results apply to the function F from Theorem 6.7. The theorem above essentially shows that when $n = 2$, this is the best possible error rate, for any shuffling-based algorithm - deterministic or random. Hence, at least for $n = 2$, the three algorithms are optimal when the component functions can possibly be non-convex. However, note that here we are only considering the dependence of the convergence rate on K . It is possible that these are not optimal, if we further take into account the dependence of the convergence rate on the combination of both n and K . Indeed, if we consider the dependence on n as well, INCREMENTAL GRADIENT DESCENT has a convergence rate of $\Omega(1/K^2)$ (Safran and Shamir, 2020), whereas the other two have a convergence rate of $\tilde{O}(1/nK^2)$ (Ahn et al., 2020).

6.6 Flipping Permutations for Faster Convergence in Quadratics

In this section, we introduce a new algorithm `FLIPFLOP`, that can improve the convergence rate of `SINGLE SHUFFLE`, `RANDOM RESHUFFLE`, and `INCREMENTAL GRADIENT DESCENT` on strongly convex quadratic functions.

The following theorem gives the convergence rate of `FLIPFLOP` WITH `SINGLE SHUFFLE`:

Assumption 6.8. $f_i(\mathbf{x})$'s are quadratic.

Theorem 6.9. *If Assumptions 6.1, 6.2, 6.3, and 6.8 hold, then running `FLIPFLOP` WITH `SINGLE SHUFFLE` for K epochs, where $K \geq 80\kappa^{3/2} \log(nK) \max\left\{1, \frac{\sqrt{\kappa}}{n}\right\}$ is an even integer, with step size $\alpha = \frac{10 \log(nK)}{\mu n K}$ gives the error*

$$\mathbb{E} \left[\left\| \mathbf{x}_n^{K-1} - \mathbf{x}^* \right\|^2 \right] = \tilde{O} \left(\frac{1}{n^2 K^2} + \frac{1}{n K^4} \right). \quad (6.4)$$

For comparison, Safran and Shamir (2020) give the following lower bound on the convergence rate of vanilla `SINGLE SHUFFLE`:

$$\mathbb{E} \left[\left\| \mathbf{x}_n^{K-1} - \mathbf{x}^* \right\|^2 \right] = \Omega \left(\frac{1}{n K^2} \right), \quad (6.5)$$

Note that both the terms in Eq. (6.4) are smaller than the term in Eq. (6.5). In particular, when $n \gg K^2$ and n is fixed as we vary K , the RHS of Eq. (6.5) decays as $\tilde{O}(\frac{1}{K^2})$, whereas the RHS of Eq. (6.4) decays as $\tilde{O}(\frac{1}{K^4})$. Otherwise, when $K^2 \gg n$ and K is fixed as we vary n , the RHS of Eq. (6.5) decays as $\tilde{O}(\frac{1}{n})$, whereas the RHS of Eq. (6.4) decays as $\tilde{O}(\frac{1}{n^2})$. Hence, in both the cases, `FLIPFLOP` WITH `SINGLE SHUFFLE` outperforms `SINGLE SHUFFLE`.

The next theorem shows that `FLIPFLOP` improves the convergence rate of `RANDOM RESHUFFLE`:

Theorem 6.10. *If Assumptions 6.1, 6.2, 6.3, and 6.8 hold, then running FLIPFLOP WITH RANDOM RESHUFFLE for K epochs, where $K \geq 55\kappa \log(nK) \max\left\{1, \sqrt{\frac{n}{\kappa}}\right\}$ is an even integer, with step size $\alpha = \frac{10 \log(nK)}{\mu n K}$ gives the error*

$$\mathbb{E} \left[\left\| \mathbf{x}_n^{K-1} - \mathbf{x}^* \right\|^2 \right] = \tilde{O} \left(\frac{1}{n^2 K^2} + \frac{1}{n K^5} \right).$$

For comparison, Safran and Shamir (2020) give the following lower bound on the convergence rate of vanilla RANDOM RESHUFFLE:

$$\mathbb{E} \left[\left\| \mathbf{x}_n^{K-1} - \mathbf{x}^* \right\|^2 \right] = \Omega \left(\frac{1}{n^2 K^2} + \frac{1}{n K^3} \right).$$

Hence, we see that in the regime when $n \gg K$, which happens when the number of components in the finite sum of F is much larger than the number of epochs, FLIPFLOP WITH RANDOM RESHUFFLE is much faster than vanilla RANDOM RESHUFFLE.

Note that the theorems above do not contradict Theorem 6.6, because for a fixed n , both the theorems above give a convergence rate of $\tilde{O}(1/K^2)$.

We also note here that the theorems above need the number of epochs to be much larger than κ , in which range Gradient Descent performs better than with- or without-replacement SGD, and hence GD should be preferred over SGD in that case. However, we think that this requirement on epochs is a limitation of our analysis, rather than that of the algorithms.

Finally, the next theorem shows that FLIPFLOP improves the convergence rate of INCREMENTAL GRADIENT DESCENT.

Theorem 6.11. *If Assumptions 6.1, 6.2, 6.3, and 6.8 hold, then running FLIPFLOP WITH INCREMENTAL GD for K epochs, where $K \geq 36\kappa \log(nK)$ is an even integer, with step size $\alpha = \frac{6 \log nK}{\mu n K}$ gives the error*

$$\mathbb{E} \left[\left\| \mathbf{x}_n^{K-1} - \mathbf{x}^* \right\|^2 \right] = \tilde{O} \left(\frac{1}{n^2 K^2} + \frac{1}{K^3} \right).$$

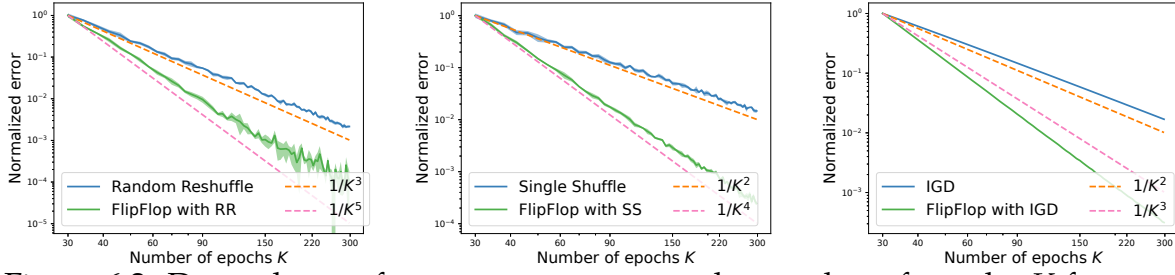


Figure 6.2: Dependence of convergence rates on the number of epochs K for quadratic functions. The figures show the median and inter-quartile range after 10 runs of each algorithm, with random initializations and random permutation seeds (note that SS and IGD exhibit extremely small variance). We set $n = 800$, so that $n \gg K$ and hence the higher order terms of K dominate the convergence rates. Note that both the axes are in logarithmic scale.

For comparison, Safran and Shamir (2020) give the following lower bound on the convergence rate of vanilla INCREMENTAL GRADIENT DESCENT:

$$\mathbb{E} \left[\left\| \mathbf{x}_n^{K-1} - \mathbf{x}^* \right\|^2 \right] = \Omega \left(\frac{1}{K^2} \right),$$

In the next subsection, we give a short sketch of the proof of these theorems.

6.6.1 Proof sketch

In the proof sketch, we consider scalar quadratic functions. The same intuition carries over to multi-dimensional quadratics, but requires a more involved analysis. Let $f_i(x) := \frac{a_i x^2}{2} + b_i x + c$. Assume that $F(x) := \frac{1}{n} \sum_{i=1}^n f_i(x)$ has minimizer at 0. This assumption is valid because it can be achieved by a simple translation of the origin (see (Safran and Shamir, 2020) for a more detailed explanation). This implies that $\sum_{i=1}^n b_i = 0$.

For the sake of this sketch, assume $x_0^1 = 0$, that is, we are starting at the minimizer itself. Further, without loss of generality, assume that $\sigma = (1, 2, \dots, n)$. Then, for the last iteration of the first epoch,

$$x_n^1 = x_{n-1}^1 - \alpha f'_n(x_{n-1}^1)$$

$$\begin{aligned}
&= x_{n-1}^1 - \alpha(a_n x_{n-1}^1 + b_n) \\
&= (1 - \alpha a_n) x_{n-1}^1 - \alpha b_n.
\end{aligned}$$

Applying this to all iterations of the first epoch, we get

$$x_n^1 = \prod_{i=1}^n (1 - \alpha a_i) x_0^1 - \alpha \sum_{i=1}^n b_i \prod_{j=i+1}^n (1 - \alpha a_j). \quad (6.6)$$

Substituting $x_0^1 = 0$, we get

$$x_n^1 = -\alpha \sum_{i=1}^n b_i \prod_{j=i+1}^n (1 - \alpha a_j). \quad (6.7)$$

Note that the sum above is not weighted uniformly: b_1 is multiplied by $\prod_{j=2}^n (1 - \alpha a_j)$, whereas b_n is multiplied by 1. Because $(1 - \alpha a_j) < 1$, we see that b_1 's weight is much smaller than b_n . If the weights were all 1, then we would get $x_n^0 = -\alpha \sum_{i=1}^n b_i = 0$, i.e., we would not move away from the minimizer. Since we want to stay close to the minimizer, we want the weights of all the b_i to be roughly equal.

The idea behind FLIPFLOP is to add something like $-\alpha \sum_{i=1}^n b_i \prod_{j=1}^i (1 - \alpha a_j)$ in the next epoch, to counteract the bias in Eq. (6.7). To achieve this, we simply take the permutation that the algorithm used in the previous epoch and flip it for the next epoch. Roughly speaking, in the next epoch b_1 will get multiplied by 1 whereas b_n will get multiplied by $\prod_{j=1}^{n-1} (1 - \alpha a_j)$. Thus over two epochs, both get scaled approximately the same.

To see more concretely, we look at the first order approximation of Eq. (6.7):

$$x_n^1 = -\alpha \sum_{i=1}^n b_i \prod_{j=i+1}^n (1 - \alpha a_j) \approx -\alpha \sum_{i=1}^n b_i (1 - \sum_{j=i+1}^n \alpha a_j) = \alpha^2 \sum_{i=1}^n b_i \sum_{j=i+1}^n a_j, \quad (6.8)$$

where in the last step above we used the fact that $\sum_{i=1}^n b_i = 0$. Now, let us see what happens in the second epoch if we use FLIPFLOP. Doing a recursion analogous to how we got Eq. (6.6),

but reversing the order of functions, we get:

$$x_n^2 = \prod_{i=1}^n (1 - \alpha a_{n-i+1}) x_0^2 - \alpha \sum_{i=1}^n b_{n-i+1} \prod_{j=i+1}^n (1 - \alpha a_{n-j+1}).$$

Carefully doing a change of variables in the equation above, we get:

$$x_n^2 = \prod_{i=1}^n (1 - \alpha a_i) x_0^2 - \alpha \sum_{i=1}^n b_i \prod_{j=1}^{i-1} (1 - \alpha a_j). \quad (6.9)$$

Note that the product in the second term in the equation above is almost complementary to the product in Eq. (6.7). This is because we flipped the order in the second epoch. This will play an important part in cancelling out much of the bias in Eq. (6.7). Continuing on from Eq. (6.9), we again do a linear approximation similar to before and substitute Eq. (6.8) (and use the fact that $x_0^2 = x_n^1$):

$$\begin{aligned} x_n^2 &\approx \left(1 - \alpha \sum_{i=1}^n a_i\right) x_0^2 + \alpha^2 \sum_{i=1}^n b_i \sum_{j=1}^{i-1} a_j \\ &\approx \left(1 - \alpha \sum_{i=1}^n a_i\right) \left(\alpha^2 \sum_{i=1}^n b_i \sum_{j=i+1}^n a_j\right) + \alpha^2 \sum_{i=1}^n b_i \sum_{j=1}^{i-1} a_j. \end{aligned}$$

We assume that α is small and hence we only focus on the quadratic terms:

$$\begin{aligned} x_n^2 &= \alpha^2 \sum_{i=1}^n b_i \sum_{j \neq i} a_j + O(\alpha^3) \\ &= \alpha^2 \left(\sum_{i=1}^n b_i \sum_{j=1}^n a_j \right) - \alpha^2 \left(\sum_{i=1}^n b_i a_i \right) + O(\alpha^3). \end{aligned}$$

Now, since $\sum_{i=1}^n b_i = 0$, we get

$$x_n^2 \approx -\alpha^2 \left(\sum_{i=1}^n b_i a_i \right) + O(\alpha^3). \quad (6.10)$$

Now, comparing the coefficients of the α^2 terms in Eq. (6.8) and Eq. (6.10), we see that the former has $O(n^2)$ terms whereas the latter has only n terms. This correction of error is

exactly what eventually manifests into the faster convergence rate of FLIPFLOP.

The main reason that the analysis for multidimensional quadratics is not as simple as the 1-dimensional analysis done above, is because unlike scalar multiplication, matrix multiplication is not commutative, and the AM-GM inequality is not true in higher dimensions (Lai and Lim, 2020; De Sa, 2020). One way to bypass this inequality is by using the following inequality for small enough α :

$$\left\| \prod_{i=1}^n (\mathbf{I} - \alpha \mathbf{A}_i) \prod_{i=1}^n (\mathbf{I} - \alpha \mathbf{A}_{n-i+1}) \right\| \leq 1 - \alpha n \mu.$$

Ahn et al. (2020) proved a stochastic version of this (see Lemma 6 in their paper). We prove a deterministic version in Lemma C.5 (in the Appendix).

6.6.2 Numerical Verification

We verify the theorems numerically by running RANDOM RESHUFFLE, SINGLE SHUFFLE and their FLIPFLOP versions on the task of mean computation. We randomly sample $n = 800$ points from a 100-dimensional sphere. Let the points be $\mathbf{x}_i$ for $i = 1, \dots, n$. Then, their mean is the solution to the following quadratic problem : $\arg \min_{\mathbf{x}} F(\mathbf{x}) = \frac{1}{n} \sum_{i=1}^n \|\mathbf{x} - \mathbf{x}_i\|^2$. We solve this problem by using the given algorithms. The results are reported in Figure 6.2. The results are plotted in a log-log graph, so that we get to see the dependence of error on the power of K .

Note that since the points are sampled randomly, INCREMENTAL GRADIENT DESCENT essentially becomes SINGLE SHUFFLE. Hence, to verify Theorem 6.11, we need ‘hard’ instances of INCREMENTAL GRADIENT DESCENT, and in particular we use the ones used in Theorem 3 in (Safran and Shamir, 2020). These results are also reported in a log-log graph in Figure 6.2.

We also tried FLIPFLOP in the training of deep neural networks, but unfortunately we did not see a big speedup there. We ran experiments on logistic regression for 1-Dimensional artificial data, the results for which are in Appendix C.8. The code for all the experiments

can be found at <https://github.com/shashankrajput/flipflop>.

6.7 Conclusion and Future Work

In this paper, we explore the theoretical limits of shuffling-based SGD for solving finite sum optimization problems. We focus on the power of good, carefully designed permutations and whether they can lead to a much better convergence rate than random. We prove that for 1-dimensional, strongly convex functions, indeed good sequences of permutations exist, which lead to a convergence rate which is exponentially faster than random permutations. We also show that unfortunately, this is not true for higher dimensions, and that for general strongly convex functions, random permutations might be optimal.

However, we think that for some subfamilies of strongly convex functions, good permutations might exist and may be easy to generate. Towards that end, we introduce a very simple technique, FLIPFLOP, to generate permutations that lead to faster convergence on strongly convex quadratics. This is a black box technique, that is, it does not look at the optimization problem to come up with the permutations; and can be implemented easily. This serves as an example that for other classes of functions, there can exist other techniques for coming up with good permutations. Finally, note that we only consider constant step sizes in this work for both upper and lower bounds. Exploring regimes in which the step size changes, *e.g.*, diminishing step sizes, is a very interesting open problem, which we leave for future work. We think that the upper and lower bounds in this paper give some important insights and can help in the development of better algorithms or heuristics. We strongly believe that under nice distributional assumptions on the component functions, there can exist good heuristics to generate good permutations, and this should also be investigated in future work.

7 SUBSEQUENT RESEARCH

There has been significant subsequent research in the field of Shuffling-Based SGD after the publication of the papers that have been presented in this thesis. Even though we were able to close the convergence gap of RR in terms of the dependence of error on the number of epochs K and dataset size n in Chapter 4 (Rajput et al., 2020), a major question still left open was the dependence of the error on the condition number κ . Mishchenko et al. (2020) and Ahn et al. (2020) were able to improve the dependence of the existing upper bounds on κ not just for RR, but also for Single Shuffle (SS) and Iterative Gradient Descent (IGD); and they also used simpler proof techniques than ours.

Another open question regarding Random Reshuffling had been regarding the ‘large epoch requirement’ - all the known upper bound results were only able to show that Random Reshuffling beats vanilla SGD (the with-replacement version) only after $K = \tilde{\Omega}(\kappa^2)$ epochs. This remained an open question until Safran and Shamir (2021) proved new lower bounds showing that in the worst case, this was indeed the case.

Lu et al. (2022) propose GraB, a Shuffling-Based algorithm¹ that provably converges faster than Random Reshuffling. Later, Cha et al. (2023) proved that GraB achieves the optimal convergence rate for any Shuffling-Based algorithm by improving the lower bounds we establish in Chapter 6 (Rajput et al., 2022).

¹Here ‘Shuffling-Based algorithm’ uses to the more general definition from Chapter 6

Part III

The Expressive Power of Modern Neural Architectures

8 LOOPED TRANSFORMERS AS PROGRAMMABLE COMPUTERS

We present a framework for using transformer networks as universal computers by programming them with specific weights and placing them in a loop. Our input sequence acts as a punchcard, consisting of instructions and memory for data read/writes. We demonstrate that a constant number of encoder layers can emulate basic computing blocks, including embedding edit operations, non-linear functions, function calls, program counters, and conditional branches. Using these building blocks, we emulate a small instruction-set computer. This allows us to map iterative algorithms to programs that can be executed by a looped, 13-layer transformer. We show how this transformer, instructed by its input, can emulate a basic calculator, a basic linear algebra library, and in-context learning algorithms that employ backpropagation. Our work highlights the versatility of the attention mechanism, and demonstrates that even shallow transformers can execute full-fledged, general-purpose programs.

8.1 Introduction

Transformers (TFs) have become a popular choice for a wide range of machine learning tasks, achieving state-of-the-art results in fields such as natural language processing and computer vision (Vaswani et al., 2017; Khan et al., 2022; Yuan et al., 2021; Dosovitskiy et al., 2020). One key reason for their success is their ability to capture higher-order relationships and long-range dependencies across tokens, through attention. This allows TFs to model contextual information and makes them effective in tasks such as machine translation and language modeling, where they have consistently outperformed other methods (Vaswani et al., 2017; Kenton and Toutanova, 2019).

This chapter presents our paper "Looped Transformers as Programmable Computers" (Giannou et al.).

Language models with billions of parameters, such as GPT-3 (175B parameters Brown et al. (2020)) and PaLM (540B parameters Chowdhery et al. (2022)), have achieved state-of-the-art performance on many natural language processing tasks. Interestingly, some of these large language models (LLMs) can also perform in-context learning, adapting to and performing a specific task, *on-the-fly*, based on a brief prompt and a few examples. The ability to perform in-context learning (ICL) arises without explicit training for it, and allows these large models to efficiently perform new tasks without requiring weight updates.

Surprisingly, through in-context learning LLMs can perform algorithmic tasks and reasoning, as demonstrated in several works including Nye et al. (2021); Wei et al. (2022c); Lewkowycz et al. (2022); Wei et al. (2022b); Zhou et al. (2022); Dasgupta et al. (2022); Chung et al. (2022). For example, Zhou et al. (2022) showed that LLMs can successfully perform addition on unseen examples when prompted with a multidigit addition algorithm and a few examples of addition. These results suggest that LLMs can apply algorithmic principles and perform pre-instructed commands on a given input at inference time, *as if interpreting natural language as code*.

Constructive arguments have demonstrated that Transformers can simulate Turing Machines with enough depth or recursive links between attention layers Pérez et al. (2021); Pérez et al. (2019); Wei et al. (2022a). This demonstrates the potential of transformer networks to precisely follow algorithmic instructions specified by the input. Yet, these constructions are more generalized and do not provide insight into how to create Transformers that can carry out particular algorithmic tasks, or compile programs in a higher-level programming language.

More specialized designs can however allow TFs to execute higher level programs. For example, in Weiss et al. (2021), the authors design a computational model and a programming language that maps simple selection and aggregation commands on indexed input tokens. This language can be used to create several interesting algorithms, such as counting tokens, sorting, creating histograms, and recognizing Dyck- k languages. Programs

written in Restricted Access Sequence Processing Language (RASP) can then be mapped into transformer networks, which typically scale in size with the size of the program.

Another line of research has demonstrated methods for selecting the weights of a Transformer model to function as an optimization algorithm for learning linear regression models on-the-fly, performing implicit training at inference time when given training data as input (Akyürek et al., 2022; von Oswald et al., 2022). These methods typically require a number of layers proportional to the number of iterations of the learning algorithm and are limited to a small set of loss functions and models.

The ability to program transformer models to emulate the abstract computation of a Turing Machine, the specialized commands of languages like RASP, and the specific algorithms of in-context learning, highlights the potential for transformer networks as versatile programmable computers. Our research aims to explore this promising prospect, uncovering how the mechanics of attention can enable the emulation of a general-purpose computer inspired by instruction-set architectures.

Our Contributions: In this chapter, we demonstrate that transformer networks can simulate complex algorithms and programs by hardcoding them with specific weights and placing them in a loop. We do this by reverse engineering attention to emulate basic computing blocks, such as edit operations on the input sequence, nonlinear functions, function calls, program counters and conditional branches. This chapter demonstrates the importance of using a single loop or recursion to connect the transformer’s output sequence back to its input, avoiding the need for a deep model.

We accomplish this by designing a transformer that can execute programs written in a generalized version of a single instruction, known as SUBLEQ(A,B,C), *i.e.*, SUBtract and branch if Less-than or Equal to zero. SUBLEQ is a single instruction language, defining a one-instruction set computer (OISC, pronounced “whisk”). SUBLEQ consists of 3 memory address operands and when executed it subtracts the value at memory address A from the

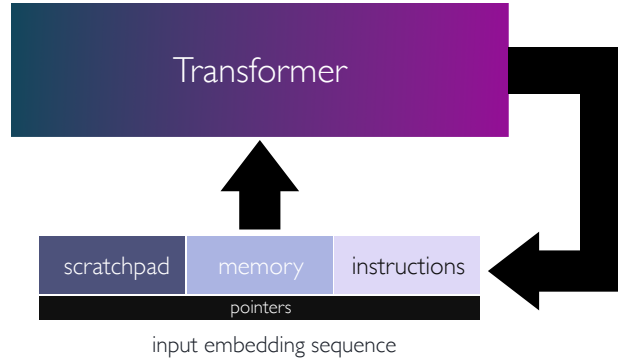


Figure 8.1: A sketch of the looped transformer architecture, where the input sequence stores the commands, memory where the data is read/written from, and a scratchpad where intermediate results are stored. The input is processed by the network and the output is used as the new input, allowing the network to iteratively update an implicit state and perform complex computations.

value at memory address B , and stores the result in B . If the result in B is less than or equal to zero, the execution jumps to address C , otherwise it proceeds to the next instruction. Programs written in SUBLEQ language use only this command, yet this single instruction is capable of defining a universal computer (Mavaddat and Parhami, 1988; Nürnberg et al., 2004).

We construct explicit transformers that implement SUBLEQ-like programs, of a more flexible single instruction which we call FLEQ which takes the form

$$\begin{aligned} \text{mem}[c] &= f_m(\text{mem}[a], \text{mem}[b]) \\ \text{if } \text{mem}[\text{flag}] &\leq 0 \\ &\quad \text{goto instruction } p \end{aligned}$$

where f_m can be selected from a set of functions (matrix multiplication/non-linear functions/polynomials/etc.),

which we can hardcode into the network. The depth of a looped transformer that can execute FLEQ programs is not dependent on the depth of the program or the number of lines of code, but rather on the depth required to implement a single FLEQ instruction, which is constant. This is achieved by running the transformer in cycles over the input

sequence, similar to how a CPU operates.

Using this framework, we demonstrate the ability to emulate a variety of functions at inference time, including a basic calculator, a basic linear algebra library (matrix transpose, multiplication, inversion, power iteration) and an in-context learning algorithm that implements backpropagation on implicit fully-connected networks. The input sequence, or the prompt, acts as a punchcard that includes the program in the form of instructions that the transformer needs to execute, while providing space for storing and processing the variables used in the program. *The transformer networks used to execute these programs are all of depth smaller or equal to thirteen, and the exact weight matrices for all these models are provided.* The following informal theorem summarizes our main findings:

Theorem 8.1 (Informal). *There exists a looped transformer with less than 13 layers that can emulate a general purpose computer (see Sec. 8.5), a basic calculator (see Sec. 8.7), numerical linear algebra methods, such as approximate matrix inverse and power iteration (see Sec. 8.8), and in-context learning algorithms, such as SGD, on neural networks (See Sec. 8.9).*

The precise size of the transformers constructed in this chapter is also summarized in Section 8.1.

	# Layers	# Heads	Formal Statement
SUBLEQ	9	2	Lemma. 8.5
Matrix Inversion	13	1	Lemma. 8.27
Power Iteration	13	1	Lemma. 8.28
SGD	13	1	Lemma. 8.31

Table 8.1: Looped transformer sizes required to successfully emulate the functionalities of a one instruction set computer (OISC), perform basic calculations, run numerical linear algebra algorithms, and in-context learning using Stochastic Gradient Descent on a neural network. The width of these networks depends on the complexity of the functions implemented, and typically range from $O(\log(\text{length\_input}) + \text{embedding\_dimension})$ to at most polynomial in the approximation error required when implementing arbitrary loss functions for in-context learning.

Our research highlights the flexibility of the attention mechanism and the importance of even a single loop making it possible to design models that can emulate complex iterative algorithms and execute general programs. It further demonstrates the ability of

transformer models to efficiently perform complex mathematical and algorithmic tasks. It is conceivable that modern transformers, such as GPT-3, utilize similar internal subroutines when performing various tasks. In a way, these models may possess the ability to elicit a specific skill or algorithm, akin to a function call, when given in-context examples and instructions. However, this hypothesis should be taken with caution, as the way we design our constructions shares no similarities with how real-world language models are trained.

We hope that our study will encourage further research into the potential of attention mechanisms, and the ability of language models to execute algorithmic instructions. Our proposed designs can aid in determining the minimal transformer network size required to perform specific algorithmic tasks. Additionally, we hope that our findings will contribute to the development of methods to enhance the capabilities of trained language models by utilizing smaller, reverse-engineered transformer networks for specific algorithmic tasks

8.2 Prior Work

Our work is inspired by the recent results on the expressive power of Transformer networks and their in-context learning capabilities.

In (Pérez et al., 2021; Pérez et al., 2019; Wei et al., 2022a) the authors explore the computational properties of Transformers establishing that they are Turing complete, meaning that they can simulate a Turing machine. The constructions typically require high/infinite precision (apart from that of Wei et al. (2022a)), and recursion around attention layers. In Yun et al. (2019a), the authors prove that given access to sufficient width/depth TFs can act as universal sequence to sequence approximators.

In Weiss et al. (2021), the authors propose a computational model for the transformer-encoder in the form of a domain-specific language called the Restricted Access Sequence Processing Language (RASP). The model maps the basic components of a TF encoder into simple primitives. Examples of tasks that could be learned by a Transformer are provided,

and the maximum number of heads and layers necessary to encode a task in a transformer are analyzed.

In a recent and related work, Lindner et al. (2023) suggests using transformer networks as programmable units and introduces a compiler called Tracr which utilizes RASP. However, the expressivity limitations and unclear Turing completeness of the language are discussed in Weiss et al. (2021); Merrill et al. (2022); Lindner et al. (2023). Our approach, in contrast, demonstrates the potential of transformer networks to serve as universal computers, enabling the implementation of arbitrary nonlinear functions and emulating iterative, non-linear algorithms. Furthermore, our framework allows the depth of our transformers *to not* scale in proportion to the lines of code that they execute, allowing the implementation of iterative algorithms, expanding the potential applications.

In Garg et al. (2022) the authors demonstrate that standard Transformers (*e.g.*, GPT-2) can be trained from scratch to perform in-context learning of linear functions and more complex model classes, such as two-layer neural networks, with performance that matches or exceeds task-specific learning algorithms. A useful element of their analysis is the fact that language is completely removed from the picture, and they perform all operations on the level of vector embeddings. This allows a higher abstraction level than using language as an input, and in fact is what also allows us to obtain our derivations.

Motivated by the above experimental work, in Akyürek et al. (2022), the authors investigate the hypothesis that TF-based in-context learners emulate standard learning algorithms implicitly at inference time. The authors provide evidence for this hypothesis by constructing transformers that implement SGD for linear models, showing that trained in-context learners closely match the predictors computed by these algorithms.

In a similar vein, von Oswald et al. (2022) argues that training Transformers on autoregressive tasks is closely related to gradient-based meta-learning formulations. The authors also provide a hard-coded weight construction showing the equivalence between data transformations induced by a single linear self-attention layer and gradient descent

on a regression loss. The authors empirically show that when training linear attention TFs on simple regression tasks, the models learned by GD and Transformers have intriguing similarities.

In Liu et al. (2022), the authors test the hypothesis that TFs can perform algorithmic reasoning using fewer layers than the number of reasoning steps, in the context of finite automata. The authors characterized “shortcut solutions” that allow shallow Transformer models to exactly replicate the computation of an automaton on an input sequence, and showed that these solutions can be learned through standard training methods. As is expected this hypothesis is only true for a certain family of automata, as the general existence of shortcut solutions would imply the collapse of complexity classes that are widely believed not to be identical.

Other experimental studies have utilized recursion in transformer architectures in a similar manner to our constructions, although in our case we only utilize a single recursive link that feeds the output of the transformer back as an input (Hutchins et al., 2022; Shen et al., 2022; Dehghani et al., 2018).

8.3 Preliminaries

The transformer architecture. Our work follows a similar problem setting as previous studies (e.g. Yun et al. (2019a); Garg et al. (2022); Akyürek et al. (2022); von Oswald et al. (2022)) in which the input sequence consists of d -dimensional embedding vectors rather than tokens. This simplifies our results without sacrificing generality, as an embedding layer can map tokens to the desired vector constructions.

The input to each layer, $\mathbf{X} \in \mathbb{R}^{d \times n}$, is a vector representation of a sequence of n tokens, where each token is a d -dimensional column. In this chapter, the terms “token” and “column” may be used interchangeably.

A transformer layer outputs $f(\mathbf{X})$, where f is defined as follows:

$$\text{Attn}(\mathbf{X}) = \mathbf{X} + \sum_{i=1}^H \mathbf{V}^i \mathbf{X} \text{softmax}(\mathbf{X}^\top \mathbf{K}^{i\top} \mathbf{Q}^i \mathbf{X}) \quad (8.1a)$$

$$f(\mathbf{X}) = \text{Attn}(\mathbf{X}) + \mathbf{W}_2 \text{ReLU}(\mathbf{W}_1 \text{Attn}(\mathbf{X}) + \mathbf{b}_1 \mathbf{1}_n^\top) + \mathbf{b}_2 \mathbf{1}_n^\top \quad (8.1b)$$

where softmax is the softmax function applied on the columns of the input matrix, *i.e.*,

$$[\text{softmax}(\mathbf{X}, \lambda)]_{i,j} = \frac{e^{\lambda X_{i,j}}}{\sum_{k=1}^n e^{\lambda X_{k,j}}},$$

where $\lambda \geq 0$ is the temperature parameter, $\text{ReLU}(x) = x \cdot 1_{x>0}$ is the ReLU activation, and $\mathbf{1}_n$ is the all ones vector of length n . We refer to the $\mathbf{K}$, $\mathbf{Q}$, and $\mathbf{V}$ matrices as the key, query, and value matrices respectively¹; the superscript i that appears on the weight matrices indicates those corresponding to the i -th attention head. Consistent with previous literature, the first equation Eq. (8.1a) represents the attention layer. We refer to the combination of attention and ReLU layers as a single transformer layer.

Iterative computation through a simple loop. In the following sections, we utilize TF networks with multiple transformer layers. Let us refer to the output of such a multilayer TF as $\text{TF}(\mathbf{W}; \mathbf{X})$, where for simplicity $\mathbf{W}$ is the collection of all weight matrices required to define such a multi-layer TF.

We use our constructions recursively, and feed the output back as an input sequence, allowing the network to perform iterative computation through a simple fixed-point like iteration. This recursive transformer is similar

Algorithm 8.1 Looped Transformer

```

1: for  $i = 1 : T$  do
2:    $\mathbf{X} \leftarrow \text{TF}(\mathbf{W}; \mathbf{X})$ 
3: end for
```

to past work on adding recursion to TF networks. We refer to these simple recursive TFs as

¹We'd like to note that typically the weight matrices are denoted as $\mathbf{W}_Q$, $\mathbf{W}_K$, $\mathbf{W}_V$ but to make notation cleaner, we use instead $\mathbf{Q}$, $\mathbf{K}$, $\mathbf{V}$.

Looped Transformers.

Feeding the output back to its input is similar to how a traditional computer processes machine code, where it continually reads/writes data in memory, by executing one instruction at a time. The input sequence $\mathbf{X}$ includes the instructions and memory. Similar to how a CPU processes each line of code in a program, the transformer network processes parts of the input sequence to perform complex computations. Like a CPU, the TF acts as a self-contained computational unit. The use of loops in this process is analogous to how CPUs operate using cycles.

While the analogy between TFs and CPUs can be entertaining, there are also many differences in implementation. It is important to keep these differences in mind and not rely too heavily on the analogy. The results obtained from using TFs as computational units do not require the analogy to be valid.

To be able to build compute boxes out of a TF network, it is crucial to format the input sequence $\mathbf{X}$ in a way that separates memory, a cache-like scratchpad, and commands.

Input sequence format. The input to our transformer network has the following abstract form:

$$\mathbf{X} = \left[\begin{array}{c|c|c} \mathbf{S} & \mathbf{M} & \mathbf{C} \\ \hline \mathbf{p}_1 & \cdots & \mathbf{p}_s & \mathbf{p}_{s+1} & \cdots & \mathbf{p}_{s+m} & \mathbf{p}_{s+m+1} & \cdots & \mathbf{p}_n \end{array} \right], \quad (8.2)$$

where $\mathbf{S}$ represents the portion of the input that serves as a “scratchpad,” $\mathbf{M}$ represents the portion that acts as memory that can be read from and written to, and $\mathbf{C}$ represents the portion that contains the commands provided by the user. The $\mathbf{p}_1, \dots, \mathbf{p}_n$ are positional encodings for the n columns, which will be described in more detail in the following paragraph, and will be used as pointers to data and instructions. The structure of our input sequence bares similarities to that of Wei et al. (2022a); Akyürek et al. (2022) that also use scratchspace, and have a separate part for the input data.

Scratchpad. The scratchpad is a crucial component of our constructions. This is the central location where the inputs and outputs of all computation are recorded. It is perhaps useful to think of this as an analogue to a CPU’s cache memory. It functions as a temporary workspace where data is copied, transformed, and manipulated in order to perform a wide variety of operations, ranging from simple arithmetic to more complex tasks such as matrix inversion. Regardless of the specific computation that is performed, the data necessary for the operation is always transferred from the memory to the scratchpad, and once the computation is completed, the data is transferred back to the memory. This allows the TF to perform the necessary calculations in a designated area, separate from other parts of the input sequence.

Memory. All the compute boxes we create require memory to perform specific actions. The memory component of the input sequence serves as a storage location for data. This data can take various forms, including scalars, vectors, and matrices, and is subject to manipulation through various operations. When computation is needed, the data is first copied from the memory to the scratchpad, where it is updated and transformed as necessary. Once the computation is complete, the updated data is then returned and copied back to the memory for future use or reference. In this way, the memory serves as a central repository for all relevant data, allowing it to be accessed and manipulated as needed.

Commands. Our framework implements a set of commands within a transformer network; these serve as instructions that guide the internal functioning of the transformer, similar to a low-level programming language. These commands include indicators for memory locations and operation directives, allowing the TF to execute complex computations and tasks in a consecutive and organized manner.

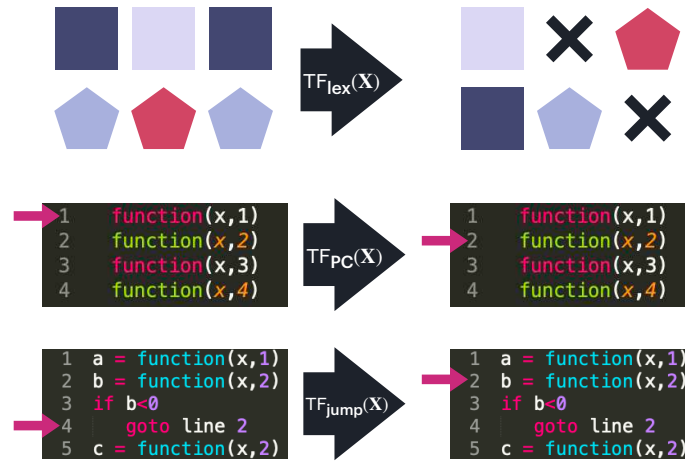


Figure 8.2: A sketch of the three transformer blocks used as building blocks to implement a small instruction-set computer. These blocks handle edits in the input sequence (such as moving or copying from one block to another), keep track of the program counter, and execute a program counter jump if a specified condition is met.

8.4 Building Transformer Blocks towards General Computation

To build general compute boxes using transformer networks, specialized compute blocks are required. These blocks will be assembled to create the desired end functionality. In this section, we highlight various operations that transformer layers can perform. These operations will serve the building blocks to create more complex routines and algorithms. These operations are designed to be interoperable with each other, leveraging the ability of attention to perform various tasks, such as producing approximate permutation matrices and approximating general functions through sigmoid activations.

In the following sections, we focus on the fundamental components necessary to emulate a general-purpose computer, reserving the examination of how attention can replicate sigmoid-based functions in the sections that follow.

8.4.1 Positional Encodings, Program Counter, and Data Pointers

To aid the transformer in locating the position of each token, each column of $\mathbf{X}$ is appended with positional encodings that is based on the column index. In this case, similar to Wei et al. (2022a), the positional encodings is the binary representation of the column index, which is appended to each column to keep the encoding dimension low, i.e., logarithmic in the sequence length. This approach to using positional encodings is slightly different from the typical method of adding them to the encodings of the input sequence. However, in this case, appending them as suffixes to the encodings allows for cleaner arguments and constructions.

In particular, the encoding for token/column indexed by i is a $\log(n)$ -dimensional ± 1 binary vector $\mathbf{p}_i \in \pm 1^{\log(n)}$, where n is the length of the input sequence. Using the standard binary representation of an integer i , meaning $i = \sum_{k=0}^{\log(n)-1} 2^k \cdot b_k$, the positional encoding vector $\mathbf{p}_i$ is set to -1 at index j if the binary representation of i has 0 at the j -th index, i.e., $b_j = 0$, otherwise it is $+1$. As a result, we have $\mathbf{p}_i^T \mathbf{p}_i = \log(n)$ and by Cauchy-Schwarz inequality, $\mathbf{p}_i^T \mathbf{p}_j < |\mathbf{p}_i| |\mathbf{p}_j| = \sqrt{\log(n)} \sqrt{\log(n)} = \log(n)$ whenever $i \neq j$, since $\mathbf{p}_i, \mathbf{p}_j$ differ in at least one coordinate.

In the applications presented, the transformer often needs to execute iterative algorithms or go through a sequence of commands. To achieve this, we utilize a program counter that iterates through the commands. The counter contains the encoding of the location where the next command is stored. Additionally, a command may have data pointers that point to the location of the data the command needs to read and write to. Both the program counter and data pointers utilize the same positional encodings as discussed in the previous paragraph. Using binary vectors as positional encodings allows us to easily increment the program counter by 1 (or any other amount) using the feed forward ReLU layers in the transformer architecture (8.1). This is formalized in the following lemma, for the proof see Theorem D.1.

Lemma 8.2. *Given two d -dimensional binary vectors representing two non-negative integers, there exists a 1-hidden layer feedforward network with ReLU activation, containing $8d$ activations in the hidden layer and d neurons in the output layer, that can output the binary vector representation of their sum, as long as the sum is less than 2^{d+1} .*

Our positional encoding scheme can also be used to point to specific data locations for reading or writing, as discussed in the following section. This is achieved by using the same binary vectors as positional encodings for both the program counter and data pointers. Furthermore, this technique for pointing to specific data locations enables the transformer to effectively read and write from/to data during the execution of the algorithm or sequence of commands that is build to implement.

8.4.2 read / write: Copying Data/Instructions to/from the Scratchpad

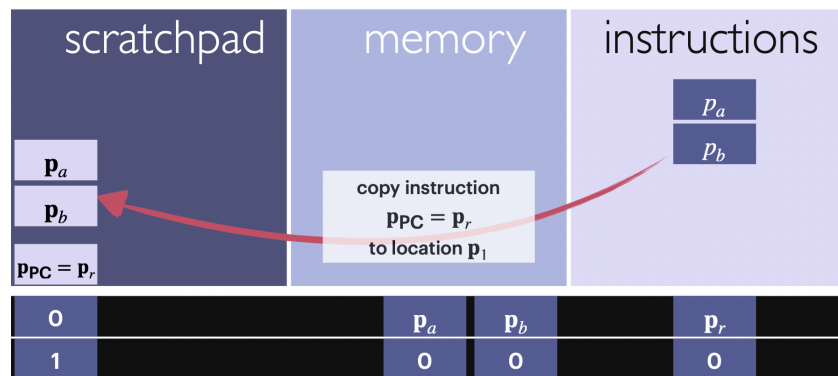


Figure 8.3: A sketch of the read operation. Arrows show command blocks being copied from the part of the input that is allocated to commands to the scratchpad. Typically an instruction is another set of pointers. Positional encodings and counters are used for tracking what is copied where.

As previously stated, the scratchpad serves as a temporary memory for storing all information needed for computation. This includes copying commands and data to it, performing computation, and writing results back to memory. This process has similarities with the copy/write mechanism developed in Akyürek et al. (2022).

The following lemma states that the command pointed to by the program counter or the data from a location specified in the current command can be copied to the scratchpad for further computation. The location of the program counter is conventionally placed right below the contents of the scratchpad, but it can be changed arbitrarily. Keeping it in a specific location throughout the entire computation helps retain a good organization of the construction.

Lemma 8.3 (read). *A transformer with one layer, one head, and width of $O(\log n + d)$, where d is the dimension of the data vectors and n is the length of the input, can read data/command vectors from the input to the scratchpad from the location pointed to by the position embedding vector in the scratchpad.*

Proof. Consider a simplified input where the scratchpad only has one column, and we have positional encodings, denoted as $\mathbf{p}_i$, that point to the location where data or commands should be copied from. In this case, the operation we want to perform is as follows:

$$\mathbf{X} = \left[\begin{array}{c|cccc} \mathbf{0} & v_2 & \cdots & v_i & \cdots \\ v_1 & \mathbf{0} & \cdots & \mathbf{0} & \cdots \\ \mathbf{p}_i & \mathbf{0} & \cdots & \mathbf{0} & \cdots \\ \mathbf{0} & \mathbf{p}_2 & \cdots & \mathbf{p}_i & \cdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \cdots \\ 1 & 0 & \cdots & 0 & \cdots \end{array} \right] \rightarrow \left[\begin{array}{c|cccc} \mathbf{0} & v_2 & \cdots & v_i & \cdots \\ v_i & \mathbf{0} & \cdots & \mathbf{0} & \cdots \\ \mathbf{p}_i & \mathbf{0} & \cdots & \mathbf{0} & \cdots \\ \mathbf{0} & \mathbf{p}_2 & \cdots & \mathbf{p}_i & \cdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \cdots \\ 1 & 0 & \cdots & 0 & \cdots \end{array} \right],$$

which moves data/command embedding vector v_i from the memory/command part of the input to the scratchpad. The first row contains the data to be read, the second row has the data written in the scratchpad, the third row contains the program counter, the fourth row contains the positional encodings, the fifth row is used by for temporary storage and the last row is just a bit that indicates whether the column is in the scratchpad or not.

We use the following key and query matrices: $\mathbf{K} = \mathbf{Q} = \begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{I} & \mathbf{I} & \mathbf{0} & \mathbf{0} \end{bmatrix}$, so that the key and query become equal to $\mathbf{KX} = \mathbf{QX} = \begin{bmatrix} \mathbf{p}_1 & \mathbf{p}_2 & \cdots & \mathbf{p}_i & \cdots \end{bmatrix}$, and hence,

$$(\mathbf{KX})^\top \mathbf{QX} = \begin{bmatrix} \mathbf{p}_1^\top \mathbf{p}_1 & \mathbf{p}_1^\top \mathbf{p}_2 & \cdots \\ \mathbf{p}_2^\top \mathbf{p}_1 & \mathbf{p}_2^\top \mathbf{p}_2 & \cdots \\ \vdots & \vdots & \vdots \\ \mathbf{p}_i^\top \mathbf{p}_1 & \mathbf{p}_i^\top \mathbf{p}_2 & \cdots \\ \vdots & \vdots & \vdots \end{bmatrix}$$

Recall that $\mathbf{p}_i$ is a $\log(n)$ -dimensional ± 1 vector such that $\mathbf{p}_i^\top \mathbf{p}_i = \log(n)$ and each $\mathbf{p}_i^\top \mathbf{p}_j \leq \log(n) - 1$ for $j \neq i$. We show in the appendix that if we apply the softmax with temperature $\lambda \geq \log \frac{n^3}{\epsilon}$, we have $\text{softmax}((\mathbf{KX})^\top \mathbf{QX})$ to be an $n \times n$ matrix of the following form

$$\begin{bmatrix} \frac{1}{2} & 0 & 0 & \cdots & \frac{1}{2} & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \frac{1}{2} & 0 & 0 & \cdots & \frac{1}{2} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & \cdots & 1 \end{bmatrix} + \epsilon \mathbf{M} = \begin{bmatrix} \frac{e_1 + e_i}{2} & e_2 & e_3 & \cdots & \frac{e_1 + e_i}{2} & \cdots \end{bmatrix} + \epsilon \mathbf{M},$$

where e_i is the i th column of the identity matrix, $\|\mathbf{M}\| \leq 1$, and ϵ is as defined in Appendix D.2. For the purpose of the proof, we ignore the error term $\epsilon \mathbf{M}$, because it can be reduced arbitrarily by increasing the temperature (it can be made precisely equal to 0, if we consider hardmax instead of softmax), and overall does not limit us from deriving arbitrarily small error bounds.

Next we set the output and value weight matrices as follows

$$\mathbf{V} = \begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{I} & \mathbf{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix}.$$

Using this, the output of the head is

$$\mathbf{X} + \mathbf{VX}_{\text{softmax}}((\mathbf{KX})^\top \mathbf{QX}) = \left[\begin{array}{c|cccc} \mathbf{0} & \mathbf{v}_2 & \cdots & \mathbf{v}_i & \cdots \\ \mathbf{v}_1 & \mathbf{0} & \cdots & \mathbf{0} & \cdots \\ \mathbf{p}_i & \mathbf{0} & \cdots & \mathbf{0} & \cdots \\ \mathbf{0} & \mathbf{p}_2 & \cdots & \mathbf{p}_i & \cdots \\ \frac{\mathbf{v}_1 + \mathbf{v}_i}{2} & \mathbf{v}_2 & \cdots & \frac{\mathbf{v}_1 + \mathbf{v}_i}{2} & \cdots \\ 1 & 0 & \cdots & 0 & \cdots \end{array} \right]$$

Each column above has the following form:

$$\begin{bmatrix} \mathbf{v}_{\text{orig}}^0 \\ \mathbf{v}_{\text{orig}}^1 \\ \mathbf{v}_{\text{orig}} \\ \mathbf{p}^{(0)} \\ \mathbf{p}^{(1)} \\ \mathbf{v}_{\text{new}} \\ b \end{bmatrix},$$

where $\mathbf{v}_{\text{orig}}^{(0)}$ and $\mathbf{v}_{\text{orig}}^{(1)}$ are the original value vectors (present in the top two row blocks)

contained in that column, $\mathbf{p}^{(0)}$ and $\mathbf{p}^{(1)}$ are the corresponding embeddings of each column, $\mathbf{v}_{\text{new}}$ is the new value, and b is the bit indicating whether the column is part of the scratchpad or not.

The feedforward layers have the following form:

$$\begin{aligned}\mathbf{v}_{\text{orig}}^{(1)} &:= \mathbf{v}_{\text{orig}}^{(1)} + \text{ReLU}(C(b-1)\mathbf{1} + 2\mathbf{v}_{\text{new}} - 2\mathbf{v}_{\text{orig}}^{(1)}) \\ &\quad - \text{ReLU}(C(b-1)\mathbf{1} - 2\mathbf{v}_{\text{new}} + 2\mathbf{v}_{\text{orig}}^{(1)}) \\ \mathbf{v}_{\text{new}} &:= \mathbf{v}_{\text{new}} - \text{ReLU}(\mathbf{v}_{\text{new}}) + \text{ReLU}(-\mathbf{v}_{\text{new}}) = \mathbf{0},\end{aligned}$$

where C is a large positive constant. The first equation is performing the operation of subtracting $\mathbf{v}_{\text{new}}$ from $\mathbf{v}_{\text{orig}}$ but only when the sum and difference of $C(b-1)\mathbf{1}$ and $\mathbf{v}_{\text{new}}$ are positive, otherwise the subtraction does not occur. The second equation is resetting the value of $\mathbf{v}_{\text{new}}$ to zero after it has been copied to $\mathbf{v}_{\text{orig}}$, where $\text{ReLU}(-\mathbf{v}_{\text{new}})$ is the rectified linear unit (ReLU) applied to the negative of $\mathbf{v}_{\text{new}}$.

It can be verified that the output of the feedforward layers would then be the desired result

$$\mathbf{X} = \left[\begin{array}{c|cccc} \mathbf{0} & \mathbf{v}_2 & \cdots & \mathbf{v}_i & \cdots \\ \mathbf{v}_i & \mathbf{0} & \cdots & \mathbf{0} & \cdots \\ \mathbf{p}_i & \mathbf{0} & \cdots & \mathbf{0} & \cdots \\ \mathbf{0} & \mathbf{p}_2 & \cdots & \mathbf{p}_i & \cdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \cdots \\ 1 & 0 & \cdots & 0 & \cdots \end{array} \right].$$

□

The next lemma explains that the vector $\mathbf{v}$ stored in the scratchpad can be copied to a designated location in memory, as specified within the scratchpad itself. This allows for the transfer of data from the scratchpad to a specific location in memory for further use or

storage.

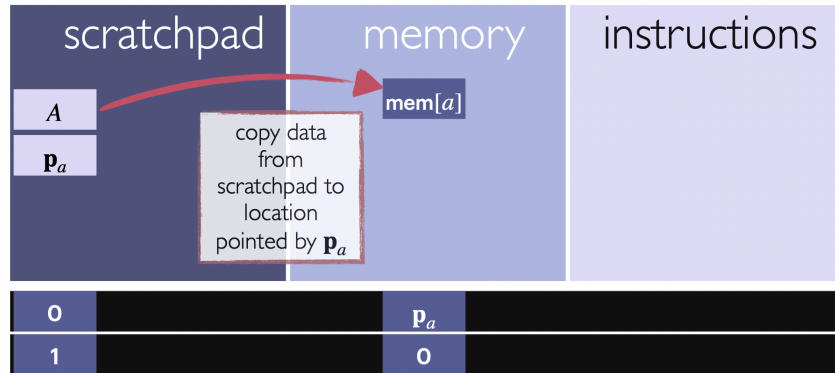


Figure 8.4: A sketch of the write operation. Arrows show data blocks being copied from the scratchpad to a designated location in the part of the input allocated for memory. Positional encodings are used for tracking the destination location and ensuring data is written at the correct memory location.

Lemma 8.4 (write). *A transformer network with a single layer, one head, and width $O(\log n + d)$, where d is the dimension of the data vectors and n is the length of the input, can effectively write a data vector stored in the scratchpad to a specific location in the input, as designated by a positional encoding vector in the scratchpad.*

Proof. We want to achieve the following operation

$$\mathbf{X} = \left[\begin{array}{c|cccc} \mathbf{0} & \mathbf{v}_2 & \cdots & \mathbf{v}_i & \cdots \\ \mathbf{v}_1 & \mathbf{0} & \cdots & \mathbf{0} & \cdots \\ \mathbf{p}_i & \mathbf{0} & \cdots & \mathbf{0} & \cdots \\ \mathbf{0} & \mathbf{p}_2 & \cdots & \mathbf{p}_i & \cdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \cdots \\ 1 & 0 & \cdots & 0 & \cdots \end{array} \right] \rightarrow \left[\begin{array}{c|cccc} \mathbf{0} & \mathbf{v}_2 & \cdots & \mathbf{v}_1 & \cdots \\ \mathbf{v}_1 & \mathbf{0} & \cdots & \mathbf{0} & \cdots \\ \mathbf{p}_i & \mathbf{0} & \cdots & \mathbf{0} & \cdots \\ \mathbf{0} & \mathbf{p}_2 & \cdots & \mathbf{p}_i & \cdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \cdots \\ 1 & 0 & \cdots & 0 & \cdots \end{array} \right],$$

The construction for this is identical to the one for read (see the proof of Theorem 8.3),

except that the feedforward layers are outputting the following:

$$\begin{aligned} \mathbf{v}_{\text{orig}}^{(0)} &:= \mathbf{v}_{\text{orig}}^{(0)} + \text{ReLU}(-C\mathbf{b}\mathbf{1} + 2\mathbf{v}_{\text{new}} - 2\mathbf{v}_{\text{orig}}^{(0)}) + \text{ReLU}(-C\mathbf{b}\mathbf{1} - 2\mathbf{v}_{\text{new}} + 2\mathbf{v}_{\text{orig}}^{(0)}) \\ \mathbf{v}_{\text{new}} &:= \mathbf{v}_{\text{new}} - \text{ReLU}(\mathbf{v}_{\text{new}}) + \text{ReLU}(-\mathbf{v}_{\text{new}}) = \mathbf{0}, \end{aligned}$$

where C is a large positive constant. The first equation updates the value of a vector $\mathbf{v}_{\text{orig}}$ in memory with the value of a vector $\mathbf{v}_{\text{new}}$ from the scratchpad. The second equation is resetting the new vector in the scratchpad to zero. It can be verified that the output of the feedforward layers would be

$$\mathbf{X} = \left[\begin{array}{c|cccc} \mathbf{0} & \mathbf{v}_2 & \cdots & \mathbf{v}_1 & \cdots \\ \mathbf{v}_1 & \mathbf{0} & \cdots & \mathbf{0} & \cdots \\ \mathbf{p}_i & \mathbf{0} & \cdots & \mathbf{0} & \cdots \\ \mathbf{0} & \mathbf{p}_2 & \cdots & \mathbf{p}_i & \cdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \cdots \\ 1 & 0 & \cdots & 0 & \cdots \end{array} \right].$$

□

8.4.3 if $\langle condition \rangle$ then goto $\langle instruction \rangle$: Conditional branching

In this subsection, we will implement a conditional branching instruction that evaluates a condition and sets the program counter to a specified location if the condition is true, or increments the program counter by 1 if the condition is false. The form of the command is as follows: if $\text{mem}[a] \leq 0$, then goto i , where $\text{mem}[a]$ is a value of some location in the memory part of the input sequence. This command has two parts: evaluating the inequality and modifying the program counter accordingly.

The first thing we do is read from $\text{mem}[a]$, as described in the previous subsection. Then, we evaluate the inequality. Let us say that “flag” is the truth value of the inequality.

Since we assume that for such conditional branching command, $\text{mem}[a]$ contains an integer, the following ReLU network can be used to compute the flag:

$$\text{flag} = 1 - \text{ReLU}(\text{mem}[a]) + \text{ReLU}(\text{mem}[a] - 1). \quad (8.3)$$

In Section 8.5.1, we consider $\text{mem}[a]$ to be vectors contain the binary ± 1 representation of integers. There we use 2's complement convention to represent negative integers. Let the vector be $[b_N \dots b_1]$, where b_N is the most significant bit and b_1 the least significant. As we explain in that section, the sign of b_N indicates whether the integer is negative or positive (The number is negative if $b_N = +1$ and non-negative otherwise). Hence, the flag is 1 if $b_N = +1$ or if all the bits are -1 (which is the case when $\text{mem}[a]$ represents the integer 0).

$$\text{flag} = \text{ReLU}(b_N) + \text{ReLU}\left(1 + N - \sum_{i=1}^N b_i\right) \quad (8.4)$$

Let the current Program Counter be $\mathbf{p}_{\text{PC}}$, which points to a given command. Thus, if flag is 1, we want the program counter to “jump” and become $\mathbf{p}_i$, else if flag is 0 the program counter will be incremented by one, and set to be $\mathbf{p}_{\text{PC}+1}$.

Consider that the simplified input currently has the following scratchpad

$$\begin{bmatrix} * & * & \dots & * & * \\ \text{flag} & 0 & \dots & 0 & 0 \\ \mathbf{p}_{\text{PC}} & 0 & \dots & 0 & 0 \\ \mathbf{p}_i & 0 & \dots & 0 & 0 \end{bmatrix},$$

where $'*'$ are inconsequential values. The incremented pointer, $\mathbf{p}_{\text{PC}+1}$, can be computed using the pointer incrementing operation that we described in the Subsection 8.4.1, using one feedforward layer of (8.1b). Then,

$$\mathbf{p}_{\text{next}} = 2\text{ReLU}(\mathbf{p}_{\text{PC}+1} - 1\text{flag}) + 2\text{ReLU}(\mathbf{p}_i - 1(1 - \text{flag})) - 1,$$

where $\mathbf{1}$ is the all ones vector. Notice that we can implement this with just the feed forward layers of Eq. (8.1b). To account for the residual connection we can add the expression $-\text{ReLU}(\mathbf{p}_{\text{PC}}) + \text{ReLU}(-\mathbf{p}_{\text{PC}})$ in the equation above.

Hence, this entire operation requires 3 feed forward layers of Eq. (8.1b), and hence 2 transformer layers. Note that to ensure that the attention layer of the transformer do not modify the input, we simply set the $\mathbf{V}$ matrix to zero in (8.1a).

8.5 Emulating a Generalized One-instruction Set Computer

8.5.1 A SUBLEQ Transformer

Mavaddat and Parhami (1988) showed that there exists an instruction such that any computer program can be translated to a program consisting of instantiation of this single instructions. A variant of such an instruction is SUBLEQ, where different registers, or memory locations are accessed. The way that SUBLEQ works is simple. It accesses two registers in memory, takes the difference of their contents and stores it back to one of the registers, and then if the result is negative it jumps to a different predefined line of code, or continues on the next instruction from the current line of code.² A computer that is built to execute SUBLEQ programs is called an One-Instruction Set Computer, and is a universal computer, *i.e.*, it is *Turing Complete*, if given access to infinite memory.

Algorithm 8.2 SUBLEQ(a, b, c)

```

1:  $\text{mem}[b] = \text{mem}[b] - \text{mem}[a]$ 
2: if  $\text{mem}[b] \leq 0$  then
3:   goto instruction  $c$ 
4: else goto next instruction
5: end if

```

²This version of the SUBLEQ instruction is a slightly restricted version of the original instruction; here we separate the memory / registers from the instructions. We show that this restriction does not make our version computationally less powerful by proving in Appendix D.3 that our version is also Turing Complete.

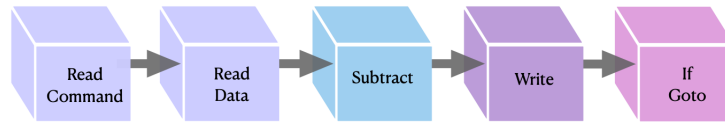


Figure 8.5: Graphical representation of the building blocks necessary to implement the OISC instruction. The first two blocks transfer the data/command to the scratchpad, the second and third implement the subtraction and store the result, while the last one implements the if goto command that completes the instruction.

The following describes the construction of a looped transformer that can execute a program written in a specific set of instructions. The transformer keeps track of the lines of code, memory locations, and a program counter, using the memory part of the input as memory registers and the command part as lines of code/instructions. The scratchpad is used to record the additions and pointers involved in each instruction, and the read, write, and conditional branch operations are utilized.

Lemma 8.5. *There exists a looped transformer architecture that can run SUBLEQ programs. This architecture has nine layers, two heads, and a width of $O(\log(n) + N)$, where n is the length of the input sequence that is proportional to the length of the program and memory used by the emulated OISC, and N is the number of bits we use to store each integer. The integers are considered to be in the range $[-2^{N-1} + 1, 2^{N-1} - 1]$*

Before we present our construction some observations are in place.

The importance of loops. The use of a loop outside the transformer is crucial as it allows the computer to keep track of the program counter and execute the instructions in the correct order. Without this loop, the size of the transformer would have to scale with the number of lines of code, making the implementation impractical. Note that the overall complexity of running a SUBLEQ program is going to scale with the number of lines of code, which is to be expected given standard complexity theoretic assumptions on the circuit depth of functions. Note however that the depth of the looped transformer itself does not scale with the size of the program.

Can we avoid the logarithmic width scaling? Finally note, that the width of the transformer scales logarithmically with the length of the program, and memory used. This is a side-effect of the bit-complexity of our positional encodings, and could be overcome by considering higher bit-complexity.

OISC as a basis for a more flexible attention-based computer. The following construction describes an implementation of a fully functioning one-instruction set computer (OISC) using a transformer architecture. The memory stores integers and the instructions are executed in a sequential manner. The key to this construction is the reverse engineering of the attention mechanism to perform read/write operations and taking full advantage of each piece of the transformer architecture, including the feedforward layers. This implementation serves as the foundation for a more general attention-based computer presented in the next subsection, where the subtraction of two contents of memory can be replaced with a general function, allowing for the implementation of arbitrary iterative algorithms.

Proof of Lemma 8.5. Looking at Algorithm 8.2, note that each instruction can be specified by just 3 indices, a , b , and c . Since we use binary representation of indices to form positional encodings and pointers, each of these indices can be represented by a $\log n$ dimensional vector. We represent each instruction by simply concatenating these embedding vectors to form a $3 \log n$ dimensional vector as follows:

$$\mathbf{c} = \begin{bmatrix} \mathbf{p}_a \\ \mathbf{p}_b \\ \mathbf{p}_c \end{bmatrix}.$$

The input then takes the following form:

The diagram illustrates the structure of the input matrix $\mathbf{X}$. It is partitioned into three main horizontal sections: a scratchpad, a memory block, and a command section. The scratchpad (columns 1 to s) contains zeros, a program counter p_{PC} , and encodings $p_{2:s}$ and $1_{2:s}$. The memory block (columns $s+1$ to $s+m$) contains a matrix $\mathbf{M}$ and encodings $p_{s+1:s+m}$ and $0_{s+1:s+m}$. The command section (columns $s+m+1$ to n) contains commands c_{s+m+1} to c_{n-1} and an EOF command, along with encodings p_{s+m+1} to p_n and 0_{s+m+1} to 0_n . Labels with arrows identify these sections: 'Block of memory' for the memory section, 'Scratchpad' for the first s columns, 'Program Counter' for p_{PC} , 'Encodings' for the bottom two rows, 'Indicator of the scratchpad' for the first row, and 'Commands' for the top row of the command section. The matrix is labeled $\mathbf{X} =$ and the equation is numbered (8.5).

$$\mathbf{X} = \begin{bmatrix} 0 & 0 & 0 & c_{s+m+1} & c_{s+m+2} & \dots & c_{n-1} & c_{EOF} \\ 0 & 0 & \mathbf{M} & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0 \\ p_{PC} & 0 & 0 & 0 & 0 & \dots & 0 & 0 \\ 0 & p_{2:s} & p_{s+1:s+m} & p_{s+m+1} & p_{s+m+2} & \dots & p_{n-1} & p_n \\ 1 & 1_{2:s} & 0_{s+1:s+m} & 0_{s+m+1} & 0_{s+m+2} & \dots & 0_{n-1} & 0_n \end{bmatrix} \quad (8.5)$$

where $c_i \in \mathbb{R}^{3 \log(n)}$, $\mathbf{M} \in \mathbb{R}^{N \times m}$ and $\mathbf{X} \in \mathbb{R}^{(8 \log(n) + 3N + 1) \times n}$. The first s columns constitute the scratchpad, the next m constitute the memory section, and the last $n - m - s$ columns contain the instructions.

The program counter, p_{PC} points to the next instruction that is to be executed, and hence it is initialized to the first instruction as $p_{PC} := p_{s+m+1}$. The contents of the memory section are N dimensional ± 1 binary vectors which represent the corresponding integers. We follow the 2's complement convention to represent the integers, described as follows. Let's say the bits representing an integer are $b_N, \dots, b_1$, with b_N being the most significant bit. Then,

1. If $b_N = -1$, then the integer is considered positive with value $\sum_{i=1}^{N-1} 2^{i-1} \frac{b_i+1}{2}$.
2. If $b_N = +1$, then the integer is considered negative with value $-2^{N-1} + \sum_{i=1}^{N-1} 2^{i-1} \frac{b_i+1}{2}$.

Step 1 - Read the instruction c_{PC} . The first thing to do is to read and copy the instruction pointed to by p_{PC} in the scratchpad. The current instruction is located at column index PC , and is pointed to by the current program counter p_{PC} . The instruction, c_{PC} consists of three pointers, each of length $\log n$. In particular we copy the elements at the location $(1 : 3 \log(n), PC)$ to the location $(3 \log(n) + 4 : 6 \log(n) + 3, 1)$. This can be done using the read operation as described in Section 8.4.2. Hence, after this operation, the input looks as

follows:

$$\mathbf{X} = \left[\begin{array}{cc|c|cccccc} \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{c}_1 & \mathbf{c}_2 & \dots & \mathbf{c}_{n-m-s} & \mathbf{c}_{\text{EOF}} \\ \mathbf{0} & \mathbf{0} & \mathbf{M} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} \\ \mathbf{c}_{\text{PC}} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} \\ \mathbf{p}_{\text{PC}} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{p}_{2:s} & \mathbf{p}_{s+1:s+m} & \mathbf{p}_{s+m+1} & \mathbf{p}_{s+m+2} & \dots & \mathbf{p}_{n-1} & \mathbf{p}_n \\ 1 & 1_{2:s} & 0_{s+1:s+m} & 0_{s+m+1} & 0_{s+m+2} & \dots & 0_{n-1} & 0_n \end{array} \right]$$

$$= \left[\begin{array}{cc|c|cccccc} \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{c}_1 & \mathbf{c}_2 & \dots & \mathbf{c}_{n-m-s-1} & \mathbf{c}_{\text{EOF}} \\ \mathbf{0} & \mathbf{0} & \mathbf{M} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} \\ \mathbf{p}_a & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} \\ \mathbf{p}_b & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} \\ \mathbf{p}_c & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} \\ \mathbf{p}_{\text{PC}} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{p}_{2:s} & \mathbf{p}_{s+1:s+m} & \mathbf{p}_{s+m+1} & \mathbf{p}_{s+m+2} & \dots & \mathbf{p}_{n-1} & \mathbf{p}_n \\ 1 & 1_{2:s} & 0_{s+1:s+m} & 0_{s+m+1} & 0_{s+m+2} & \dots & 0_{n-1} & 0_n \end{array} \right]$$

This step can be done in one layer.

Step 2 - Read the data required by the instruction. We need to read the data that the columns a, b contain. To do so, we again use the read operation on the pointers $\mathbf{p}_a, \mathbf{p}_b$. Note that we need two heads for this operation, one each for reading a and b . The resulting

output sequence looks like

$$\mathbf{X} = \left[\begin{array}{cc|c|cccccc} \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{c}_1 & \mathbf{c}_2 & \dots & \mathbf{c}_{n-m-s-1} & \mathbf{c}_{\text{EOF}} \\ \mathbf{0} & \mathbf{0} & \mathbf{M} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} \\ \text{mem}[a] & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} \\ \text{mem}[b] & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} \\ \mathbf{p}_a & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} \\ \mathbf{p}_b & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} \\ \mathbf{p}_c & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} \\ \mathbf{p}_{\text{PC}} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{p}_{2:s} & \mathbf{p}_{s+1:s+m} & \mathbf{p}_{s+m+1} & \mathbf{p}_{s+m+2} & \dots & \mathbf{p}_{n-1} & \mathbf{p}_n \\ 1 & 1_{2:s} & 0_{s+1:s+m} & 0_{s+m+1} & 0_{s+m+2} & \dots & 0_{n-1} & 0_n \end{array} \right]. \quad (8.6)$$

This step can be done in one layer.

Step 3 - Perform subtraction. Let x denote a column of the input $\mathbf{X}$. Let it have the following structure:

$$\mathbf{x} = \begin{bmatrix} * \\ * \\ \mathbf{b}_r \\ \mathbf{b}_s \\ * \\ * \\ * \\ * \\ * \\ * \end{bmatrix},$$

where each entry above represents the corresponding column element of the matrix $\mathbf{X}$ in (8.6). Thus, $\mathbf{b}_r = \text{mem}[a]$, $\mathbf{b}_s = \text{mem}[b]$ for the first column, and $\mathbf{b}_r = \mathbf{b}_s = \mathbf{0}$ otherwise.

Hence, to perform $\mathbf{b}_{s-r}$, we first need to compute the binary representation of $-r$, which is $\mathbf{b}_{-r}$, and then simply add it to $\mathbf{b}_s$. To compute $\mathbf{b}_{-r}$, which is the 2's complement of $\mathbf{b}_r$, we just need to flip the bits of $\mathbf{b}_r$ and add 1. Bit flipping a ± 1 bit can be done with a neuron simply as $b_{\text{flipped}} = 2 * \text{ReLU}(-b) - 1$. For adding 1, we can use Theorem D.1. Hence, each of these operations can be done using 1 ReLU layer of width $O(N)$, and so we need 2 transformer layers to perform this (Here we make the intermediate attention layers become the identity mapping by setting their value matrices to $\mathbf{0}$). Finally, we need one more ReLU layer to add $\mathbf{b}_s$ to $\mathbf{b}_{-r}$, hence bringing the total to 3 transformer layers.

This results in the following:

$$\mathbf{X} = \left[\begin{array}{cc|c|cccccc} \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{c}_1 & \mathbf{c}_2 & \dots & \mathbf{c}_{n-m-s-1} & \mathbf{c}_{\text{EOF}} \\ \mathbf{0} & \mathbf{0} & \mathbf{M} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} \\ \text{mem}[b] - \text{mem}[a] & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} \\ \mathbf{p}_a & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} \\ \mathbf{p}_b & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} \\ \mathbf{p}_c & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} \\ \mathbf{p}_{\text{PC}} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{p}_{2:s} & \mathbf{p}_{s+1:s+m} & \mathbf{p}_{s+m+1} & \mathbf{p}_{s+m+2} & \dots & \mathbf{p}_{n-1} & \mathbf{p}_n \\ 1 & 1_{2:s} & 0_{s+1:s+m} & 0_{s+m+1} & 0_{s+m+2} & \dots & 0_{n-1} & 0_n \end{array} \right]$$

Note that since this can be done in the feedforward layers of the previous step, this does not require an additional layer.

Step 4 - Write the result back to memory. Writing $\text{mem}[b] - \text{mem}[a]$ back to location b can be done using the pointer $\mathbf{p}_b$ and the set of embeddings and applying the write operation

described in Section 8.4.2. This operation requires one layer.

Step 5 - Conditional branching. We first use Eq. (8.4) as described in Section 8.4.3 to create the flag, which is 1 if $\text{mem}[b] - \text{mem}[a] \leq 0$ and 0 otherwise. This can be done using the Eq. (8.1b) of the transformer. Thus, we have

$$\mathbf{X} = \left[\begin{array}{cc|c|cccccc} \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{c}_1 & \mathbf{c}_2 & \dots & \mathbf{c}_{n-m-s-1} & \mathbf{c}_{\text{EOF}} \\ \mathbf{0} & \mathbf{0} & \mathbf{M} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} \\ \text{flag} & 0 & 0 & 0 & 0 & \dots & 0 & 0 \\ \mathbf{p}_a & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} \\ \mathbf{p}_b & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} \\ \mathbf{p}_c & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} \\ \mathbf{p}_{\text{PC}} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{p}_{2:s} & \mathbf{p}_{s+1:s+m} & \mathbf{p}_{s+m+1} & \mathbf{p}_{s+m+2} & \dots & \mathbf{p}_{n-1} & \mathbf{p}_n \\ 1 & 1_{2:s} & 0_{s+1:s+m} & 0_{s+m+1} & 0_{s+m+2} & \dots & 0_{n-1} & 0_n \end{array} \right] \quad (8.7)$$

This operation requires one layer.

Next we use the construction described in Section 8.4.3 to choose, depending on the value of the flag, whether we want to increment the current program counter or we want to jump in the command c . Similar to Section 8.4.3, this step needs 2 layers of transformers.

Step 6 - Error Correction. Note that some of the steps above we incur some error while reading and writing due to the fact that we are using softmax instead of hardmax. This error can be made arbitrarily small by increasing the temperature of the softmax. In this step, we push the error down to zero. Note that all the elements of $\mathbf{X}$ can only be one of $\{-1, 0, 1\}$, with some additive error from reads and writes as explained before. Assume that the temperature is set high enough that the error is at most $\epsilon < 0.5$. Then, a noisy bit b

can be fixed using the following ReLU:

$$b_{\text{noiseless}} = \frac{1}{1 - 2\epsilon} (\text{ReLU}(b + 1 - \epsilon) - \text{ReLU}(b + \epsilon)) \\ + \frac{1}{1 - 2\epsilon} (\text{ReLU}(b - \epsilon) - \text{ReLU}(b - 1 + \epsilon)) - 1.$$

This operation can be done with a single layer of transformer.

Step 7 - Program Termination. The special command c_{EOF} is used to signal the end of a program to the transformer. This command is made up of three encodings: p_{s+1} , p_{s+2} , and p_n . The first encoding, p_{s+1} , points to the first entry in the memory, which we hard-code to contain the value 0. The second encoding, p_{s+2} , points to the second entry in the memory, which is hard-coded to contain the value -1 . The third encoding, p_n , points to itself, signaling the end of the program and preventing further execution of commands. Hence, on executing this command, the next command pointer is set to point to this command again. This ensures that the transformer maintains the final state of the input.

- For this, we ensure that the last instruction in each program is c_{EOF} , and that $\text{mem}[s+1] = 0$ and $\text{mem}[s+2] = -1$.
- For this case $a = s+1$, $b = s+2$, and $c = n$.
- The memory is updated with the value $\text{mem}[b] = \text{mem}[b] - \text{mem}[a]$. Since $\text{mem}[a] = 0$ here, the memory remains unchanged.
- Since $\text{mem}[b] \leq 0$ here, the branch is always true and thus the pointer for the next instruction is again set to point to c_{EOF} .

□

8.5.2 FLEQ: A More Flexible Attention-based Computer

In this section, we introduce FLEQ, a generalization of SUBLEQ that defines a more flexible reduced-instruction set computer. This implied set of additional instructions is based on a more advanced version of SUBLEQ that allows for the implementation of multiple functions within the same transformer network. This is achieved by generalizing the previous OISC construction to include not just addition of registers, but any function from a set of M predefined functions implementable by a transformer network. In the following, we use the term FLEQ to refer interchangeably to the instruction, the language, and the attention-based computer it defines.

The design of FLEQ allows for the implementation of complex and sophisticated algorithms by generating more general functions beyond simple subtraction, such as matrix multiplication, computation of square roots, activation functions, etc. This not only increases the flexibility of the system, but also makes it possible to implement nonlinear computations, linear algebra calculations, and iterative optimization algorithms for in-context learning while containing the length of the corresponding programs.

Definition 8.6. Let $\mathcal{T}_i$ be a transformer network of the form (8.1) with l_i -layers, h_i -heads and dimensionality r . We call this a “**transformer-based function block**” if it implements a function $f(\mathbf{A}, \mathbf{B})$ where the input and output sequence format is assumed to be the following: $\mathbf{A} \in \mathbb{R}^{d_h \times d_w}$ is assumed to be provided in the first set of d columns (columns 1 to d) and $\mathbf{B} \in \mathbb{R}^{d_h \times d_w}$ the second set of d columns (columns $d + 1$ to $2d$); after passing the input through the l_i layers, the output of $f(\mathbf{A}, \mathbf{B}) \in \mathbb{R}^{d_h \times d_w}$ is stored in the third d columns (columns $2d + 1$ to $3d$), where d is the maximum size that the input could have and it is a constant that we determine. Note that $d_h, d_w \leq d$. Finally, the sequence length of the block is $s \geq 3d$. Similarly to d , s is a predetermined constant.

The parameters $\mathbf{A}, \mathbf{B}$ can be scalars, vectors or matrices as long as they can fit within a $d \times d$ matrix. Hence, the above definition is minimally restrictive, with the only main constraint being the input and output locations. More details about the input and output

requirements will be explained towards the end of this subsection.

Theorem 8.7. *Given M different transformer-based function blocks $\mathcal{T}_1, \dots, \mathcal{T}_M$, there exists a transformer $\mathcal{T}$ of the form (8.1) with number of layers $9 + \max\{l_1, \dots, l_M\}$, a number of $\sum_{i=1}^M h_i$ heads, and dimensionality $O(Md + \log n)$ such that running it recurrently T times can run T instructions of any program where each instruction is $\text{FLEQ}(a, b, c, m, \text{flag}, p, d_h, d_w)$, and executes the following:*

$$\text{mem}[c] = f_m(\text{mem}[a], \text{mem}[b]) \quad ; \quad \text{if } \text{mem}[\text{flag}] \leq 0 \text{ goto instruction } p \quad (8.8)$$

Here n is the total length of the program and we assume that $\text{mem}[\text{flag}]$ is an integer. The parameters d_h, d_w are explained in Theorem 8.8 below.

Remark 8.8. *Note that, the transformer $\mathcal{T}$ contains M transformer-based function blocks and each one may use different input parameters. We thus define with d the max length that each of the parameters $\mathbf{A}, \mathbf{B}, \mathbf{C}$ (stored in locations a, b, c) as in Theorem 8.6 can have; this is a global constant and it is fixed for all the different instances that we can create. Now, d_h, d_w refer to the maximum dimension that the parameters can have in a specific instance of the transformer $\mathcal{T}$; the rest of the columns $d - d_w$ and rows $d - d_h$ are set to zero.*

The proof of this theorem can be found in Appendix D.4. Below we explain some of our design choices.

Execution cycle of the unified attention-based computer. In each iteration of the looped transformer, one instruction is fetched from the set of instructions in the input according to the program counter. The instruction is then copied to the scratchpad. Depending on the function to be implemented, a different function block location is used to locally record the results of that function. Once the result is calculated, it is copied back to a specified memory location provided by the instruction. The execution cycle is similar to the one-instruction set computer (OISC) in the previous section, with the main difference being that for each

instruction, we can choose from a pre-selected list of functions that take inputs in the form of arbitrary arrays of numbers, such as matrices, vectors, and scalars.

The format of the input sequence. In Fig. 8.6, we illustrate the input X to our looped transformer, which can execute a program written as a series of FLEQ instructions. Note that X is divided into three sections: Scratchpad, Memory, and Instructions. As in the left bottom part of Fig. 8.6, we allocate a separate part of the scratchpad for each of the M functions that are internally implemented by the transformer. For example, if we have matrix multiplication and element-wise square root as two functions, we would allocate a different function block for each one.

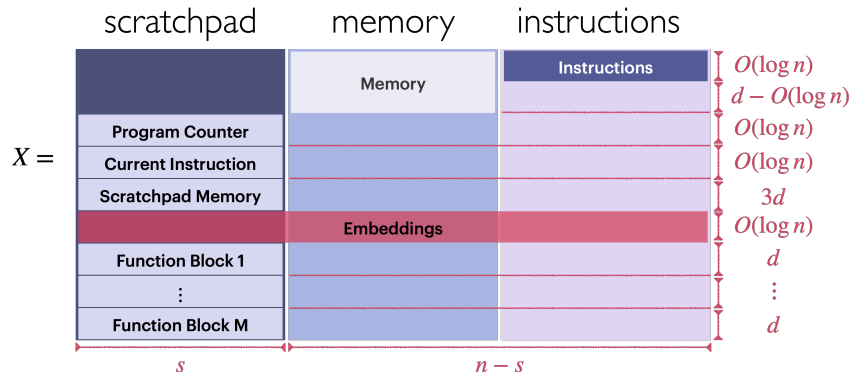
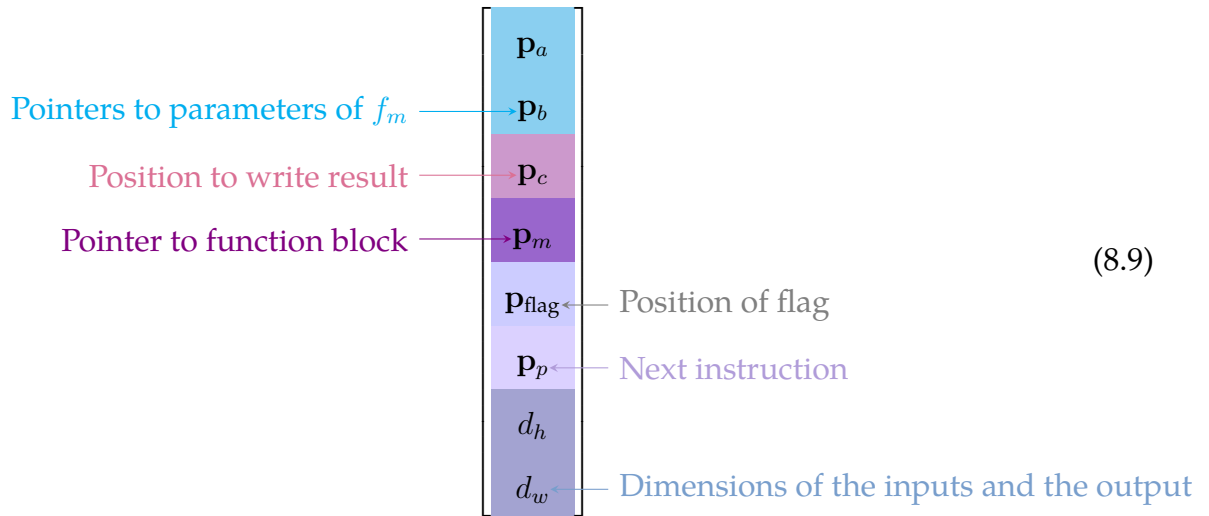


Figure 8.6: The structure of input X , to execute FLEQ commands.

This design may not be the most efficient, but our goal is to demonstrate the possibilities of looped transformers. Additionally, since the number of different functions is typically small in the applications we have in mind, the design does not significantly increase in size. The choice to reserve different function blocks for each predefined function is for convenience, as it allows for separate treatment of functions without worrying about potentially overlapping results. We believe that a design with a single function block is feasible, but it would significantly complicate the rest of the transformer construction.

Instruction format. The instruction in Theorem 8.7 is essentially a composition of the following two components: the function call to f_m and the conditional branching (if ...

goto ...). The instruction, located at the top right side of Fig. 8.6 contains the following components:



The goal of each positional encoding vector in Eq. (8.9) is to point to the corresponding space of the input where each component required by the instruction is located. To be specific, p_a and p_b point to the locations that the inputs a and b are located, p_c points to the location to which we will record the final result of the function f_m . Similarly, p_m points to the function block in the scratchpad that the intermediate computations required for f_m are recording, p_{flag} points to the variable that we check if it is non-positive (the result is used for conditional branching), and p_p points to the address of the line of code that we would jump if the variable in pointed by p_{flag} is non-positive.

Execute a function; Jump to command. Recall that the first four parameters (a, b, c, m) of FLEQ, as well as the last two (d_h, d_w) are related to the implementation of the function block, while the other two (flag, p) are related with the conditional branching. Since there is no overlap between the two components of each instruction, it is possible to use each of these components independently. By having a fixed location flag_0 where $\text{mem}[\text{flag}_0]$ is always set to 1, we can have the simpler command $\text{FLEQ}(a, b, c, m, \text{flag}_0, p, d_h, d_w)$ which

implements

$$\text{mem}[c] = f_m(\text{mem}[a], \text{mem}[b]).$$

Further, by having fixed locations a_0, b_0, c_0 which are not used elsewhere in the program, and hence inconsequential, we can have the simpler command

$\text{FLEQ}(a_0, b_0, c_0, m, \text{flag}, p, d_h, d_w)$ which implements

$$\text{if mem}[\text{flag}] \leq 0 \text{ goto instruction } p.$$

Using this, we get the following corollary:

Corollary 8.9. *The Unified Attention Based Computer presented in Theorem 8.7 can run programs where each instruction can be **either** of the following two simple instructions:*

- $\text{mem}[c] = f_m(\text{mem}[a], \text{mem}[b])$
- $\text{if mem}[\text{flag}] \leq 0 \text{ goto instruction } p$

Format of Transformer-Based Function Blocks. Recall that each function block is located at the bottom left part of the input X , as shown in Fig. 8.6. Each transformer-based function block is expected to operate using the following format of the input:

- The number of rows in the input is r , while the number of columns is s and $s \geq 3d$. Here s will dictate the total maximum number of columns that any transformer-based function block needs to operate. The reason that s might be larger than $3d$ has to do with the fact that some blocks may need some extra scratchpad space to perform some calculations.
- The function block specifies the dimensions of input and output. Say they are $d_h \times d_w$, where $d_h, d_w \leq d$. These will be part of the instruction which calls this function inside the FLEQ framework, as in (8.9).

- Suppose each function block has two inputs ($\mathbf{A} \in \mathbb{R}^{d_h \times d_w}$ and $\mathbf{B} \in \mathbb{R}^{d_h \times d_w}$) and one output $f(\mathbf{A}, \mathbf{B}) = \mathbf{C} \in \mathbb{R}^{d_h \times d_w}$. As in (8.10), the function block is divided into four parts: (1) the first input $\mathbf{A}$ is placed in the first d_h rows and the first d_w columns, (2) the second input $\mathbf{B}$ is placed in the first d_h rows and the columns $d + 1 : d + d_w$, (3) the output $f(\mathbf{A}, \mathbf{B}) = \mathbf{C}$ is in the first d_h rows and the columns $2d + 1 : 2d + d_w$ columns and 4) the rest $s - 3d$ column used as scratchpad space for performing necessary calculations. Note that the unused columns are set to zero.
- The last $r - d_h$ rows can be used by the transformer-based function block in any way, *e.g.*, to store any additional positional encodings.

We put the format of the input of each *transformer-based function block* in (8.10). The first input $\mathbf{A} = [z_a^1, \dots, z_a^{d_w}]$ of the function is zero padded and stored in the first d columns. Similarly, the second input $\mathbf{B} = [z_b^1, \dots, z_b^{d_w}]$ is stored in the next d columns. The output/result of the function block $\mathbf{C} = [z_c^1, \dots, z_c^{d_w}]$ is located in the next d columns while we have some extra $s - 3d$ columns which can be used as scratchpad.

$$\begin{array}{c}
 \text{Input A} \quad \text{Input B} \quad \text{Output } C = f(A, B) \\
 \left[\begin{array}{cccc|cccc|cccc|cccc}
 z_a^1 & \dots & z_a^{d_w} & 0 & z_b^1 & \dots & z_b^{d_w} & 0 & z_c^1 & \dots & z_c^{d_w} & 0 & \dots & 0 \\
 * & \dots & * & * & * & \dots & * & * & * & \dots & * & * & \dots & *
 \end{array} \right] \quad (8.10)
 \end{array}$$

Let us consider the case where we wish to multiply a matrix $\mathbf{A} \in \mathbb{R}^{d \times d}$ with a vector $\mathbf{b} \in \mathbb{R}^{d \times 1}$. The resulting output matrix would look as follows:

$$\left[\mathbf{A} \mid \mathbf{b} \mid \mathbf{0} \mid \mathbf{A}^\top \mathbf{b} \mid \mathbf{0} \mid \mathbf{0} \right].$$

Computational concerns: Do we need full attention? In our construction, the computational complexity of each layer depends on the number of embedding vectors that each part of the input has to attend to. Typically, this is quite sparse, as only a few of them need global attention. In our specific construction, only the columns within the scratchpad require

global attention. By focusing only on these columns, we can reduce the computational complexity of the attention mechanism from $O(n^2d)$ to $O(nd)$, where n is the number of input sequences, d is the dimension of the embedding vectors.

This reduction in computational complexity is achieved by limiting the attention mechanism to only the columns within the scratchpad, which helps to improve the overall efficiency of the model. Additionally, since the computational complexity grows linearly with the number of input sequences, rather than quadratically, it enables us to scale the model to handle larger input sequences.

8.6 Functions in the Unified Template Form

In this section, we demonstrate how to implement a variety of nonlinear functions and basic linear algebra operations using transformers. These techniques will be crucial in the construction of iterative algorithms in the following sections. Each transformer-based function block in this section fits in our unified template in terms of input/output parameters' locations. We note here that each transformer-based function block might have its own positional encodings used to transfer the output in the correct place or perform some read/write operations and they are part of the design of the block.

8.6.1 Encoding Non-linear Functions within the Attention Mechanism

One key ingredient of our constructions is encoding various functions within the attention mechanism. We do this by forcing the softmax to act as a sigmoid function and by storing multiple coefficients in the query and value weight matrices. As far as we know, this is the first work that shows how general non-linear functions can be emulated by attention layers. This allows us to create linear combinations of sigmoids that can be accessed by an indicator vector in the input. Our analysis is based on the result of Barron (1993b) which we present below.

Definition 8.10. Let $\Gamma_{C,B}$ be the set of functions defined in a bounded domain B , $f : B \rightarrow \mathbb{R}$, $B \subseteq \mathbb{R}^d$ with a proper extension to $\mathbb{R}^d$ such that they have C bounded Fourier integral, i.e., $\int \sup_{x \in B} |w \cdot x| F(dw) \leq C$ holds where $F(dw)$ is the magnitude of the Fourier distribution.

Definition 8.11. Given $\tau > 0, C > 0$ and a bounded set B , let

$$G_{\sigma,\tau} = \{\gamma\sigma(\tau(\mathbf{a}^T \mathbf{x} + b)) : |\gamma| \leq 2C, \|\mathbf{a}\|_B \leq 1, |b| \leq 1\}$$

where $\|\mathbf{a}\|_B = \sup_{\mathbf{x} \in B} \{\mathbf{x}^T \mathbf{a}\}$ and σ is the sigmoid function, i.e., $\sigma(x) = \frac{1}{1+e^{-x}}$.

Theorem 8.12 (Theorem 3 in Barron (1993b)). Every function $f \in \Gamma_{C,B}$ with $f(0) = 0$ and can be approximated by a linear combination of sigmoids $f_i \in G_{\sigma,\tau}$, $i = 1, \dots, m$. If $\tau \geq m^{1/2} \ln m$ the error scales as

$$\left| f(\mathbf{x}) - \sum_{i=1}^m f_i(\mathbf{x}) \right| \leq O\left(\frac{1}{m^{1/2}}\right), \mathbf{x} \in B$$

To encode N different functions, we use the index $j \in [N]$ and write $c_{ji}, \mathbf{a}_{ji}$ for the coefficients of the sigmoids that approximate them or

$$f_j(\mathbf{x}) = \sum_{i=1}^m c_{ji} \sigma(\mathbf{x}^T \mathbf{a}_{ji}) \text{ for } j = 1, \dots, N$$

We here note that the terms τ, b can be incorporated in the term $\mathbf{a}_{ji}$ by adding an extra coefficient of 1 in $\mathbf{x}$ and multiplying everything with τ .

We are now able to present the lemma on approximating functions using transformer blocks, in a format that is consistent with the FLEQ design outlined in the previous section.

Lemma 8.13. Fix $\epsilon > 0$ and consider an input of the form

$$\mathbf{X} = \left[\begin{array}{cc|cc|cc} \mathbf{e} & \mathbf{0} & \mathbf{x} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{p}_{2d+1} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{p}_1 & \mathbf{p}_{2:d} & \mathbf{0} & \mathbf{p}_{d+2:2d} & \mathbf{p}_{2d+1} & \mathbf{p}_{2d+2:3d} \\ 0 & 0_{2:d} & 1 & 0_{d+2:2d} & 0 & 0_{2d+2:3d} \end{array} \right].$$

where d is chosen according to the FLEQ construction from the previous section and N is the number of functions we encode. $\mathbf{e} = \mathbf{e}_j \in \mathbb{R}^N$ is an indicator vector signifying the function we wish to execute. Then there exists a transformer-based function block with 3 layers, m heads and dimensionality $r = 2 \log(d) + d + 1 = O(d)$ such that

$$f(\mathbf{X}) = \left[\begin{array}{cc|cc|cc} * & * & * & * & \sum_{i=1}^m c_{ji} \sigma(\mathbf{x}^T \mathbf{a}_{ji}) + \epsilon & * \\ \mathbf{0} & \mathbf{0} & \mathbf{x} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{p}_{2d+1} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{p}_1 & \mathbf{p}_{2:d} & \mathbf{0} & \mathbf{p}_{d+2:2d} & \mathbf{p}_{2d+1} & \mathbf{p}_{2d+2:3d} \\ 0 & 0_{2:d} & 1 & 0_{d+2:2d} & 0 & 0_{2d+2:3d} \end{array} \right]$$

where $*$ denoted inconsequential values that will be ignored downstream. This implies that arbitrary function $g \in \Gamma_{C,B}$ can be well approximated by attention layers.

Remark 8.14. Notice that in this case we don't use any extra scratchpad space and thus $s = 3d$; however if this function block was to be used with another one that needs $s > 3d$ scratchpad space, we would simply zero pad the input of Theorem 8.13 and ignore these columns. The same holds for the rest of the transformer-based function blocks and we will not mention it from now on.

In the expression $\sum_{i=1}^m c_{ji} \sigma(\mathbf{x}^T \mathbf{a}_{ji})$, the number head is equal to the number of terms we need. We show in the appendix that we can actually encode these m terms in the dimension of the transformer architecture with just one head (See Theorem D.4). The choice of which

result to use can depend on the specific design and can affect both accuracy and efficiency of the implemented transformer network.

The proof of this Lemma is given in Appendix D.1.2.

8.6.2 Matrix Transposition and Multiplication by Linearizing the Softmax

We assume that a $d \times d$ matrix $\mathbf{A}$ in the input $\mathbf{X}$ is represented by a sequence of length d , and each of these d columns has d rows. While this representation has the advantage that it is well suited for the matrix multiplication operation (as we will see in the next sub-section), a vectorized form of the matrix is more suited to create transpose. This is how we implement the transpose; we first vectorize the matrix $\mathbf{A}$, then with a fixed permutation of the columns we create its vectorized version of a transpose.

Lemma 8.15. *Fix $\epsilon > 0$ and consider an input of the following form*

$$\mathbf{X} = \left[\begin{array}{c|c|c|c|c} \mathbf{A} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{P}_{1:d} & \mathbf{P}_{1:d} & \mathbf{P}_{1:d} & \dots & \mathbf{P}_{1:d} \\ \mathbf{P}'_1 & \mathbf{P}'_2 & \mathbf{P}'_3 & \dots & \mathbf{P}'_d \end{array} \right].$$

where $\mathbf{A} \in \mathbb{R}^{d \times d}$; then there exists transformer-based function block with 4 layers, 1 head and dimensionality $r = 2d + 2 \log d = O(d)$ that outputs the following matrix

$$\mathbf{X} = \left[\begin{array}{c|c|c|c|c} \mathbf{A}' & \mathbf{A}' & \mathbf{A}' & \dots & \mathbf{A}' \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{P}_{1:d} & \mathbf{P}_{1:d} & \mathbf{P}_{1:d} & \dots & \mathbf{P}_{1:d} \\ \mathbf{P}'_1 & \mathbf{P}'_2 & \mathbf{P}'_3 & \dots & \mathbf{P}'_d \end{array} \right].$$

where $\mathbf{A}' = \mathbf{A}^\top + \epsilon \mathbf{M}$, for some $\|\mathbf{M}\| \leq 1$. The error ϵ depends on the choice of the temperature λ ,

as it is a consequence of the *read/write* operations.

In order for matrix multiplication to fit in our unified template, we need to show for example for the result of $\mathbf{A}^\top \mathbf{B}$, where $\mathbf{A} \in \mathbb{R}^{k \times m}$ and $\mathbf{B} \in \mathbb{R}^{k \times n}$ with $k, m, n < d$ we can achieve the following:

$$\left[\begin{array}{cc|cc|cc} \mathbf{A} & \mathbf{0} & \mathbf{B} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{array} \right] \rightarrow \left[\begin{array}{cc|cc|cc} * & * & * & * & \mathbf{A}^\top \mathbf{B} & * \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{array} \right]$$

The idea we leverage is the linearization of the softmax, *i.e.*, for a column vector $\mathbf{z} = [\mathbf{x}, C]$ for some large constant C we have that

$$\text{softmax}(\mathbf{z}) = [\mathbf{x} + \epsilon, *]$$

The error ϵ is controlled by the constant C .

Lemma 8.16. *Let $\mathbf{A} \in \mathbb{R}^{k \times m}$ and $\mathbf{B} \in \mathbb{R}^{k \times n}$; then for any $\epsilon > 0$ there exists a transformer-based function block with 2 layers, 1 head and dimensionality $r = O(d)$ that outputs the multiplication $\mathbf{A}^\top \mathbf{B} + \epsilon \mathbf{M}$, for some $\|\mathbf{M}\| \leq 1$.*

The implementation of $\mathbf{B}^\top \mathbf{A}$, $\mathbf{A}^\top \mathbf{A}$ and $\mathbf{B}^\top \mathbf{B}$ are simple corollaries of the lemma presented above and we will freely use them in the subsequent sections. In Appendix D.1.2, we provide the exact form of the input $\mathbf{X}$ for implementing matrix transposition/multiplication, as well as the proof of the corresponding Lemmas.

8.6.3 Advantage of attention over fully-connected networks

It is possible to implement the functions and overall lexicographic functionality presented in previous sections using fully connected networks, as they are also universal function approximators. However, it is easy to demonstrate a depth separation between attention-based networks and fully connected networks. For example, to compute simple functions

like polynomials of x (e.g., x^2), a ReLU network with a depth proportional to $\log(1/\epsilon)$ is required, where ϵ is the quality of approximation, e.g., as showed in (Perekrestenko et al., 2018). In contrast, we have shown how x^2 can be implemented in essentially 2 layers. This simple depth separation argument highlights the constant vs scaling depth required for several functionalities in fully connected networks versus attention-based networks. It is important to note that although these constructions are easy to demonstrate their existence, constructing them is not straightforward. In this work, we provide hardcoded attention layers that precisely do that, making it easier to implement these functionalities in practice.

8.7 A Basic Calculator

We show that the FLEQ transformer introduced in Section 8.5.2, can be used to build a simple calculator. This transformer consists of six transformer-based function blocks that implement addition, subtraction, multiplication, percentage, division and square root. The formal statement is written as below.

Theorem 8.17. *There exists a transformer with 12 layers, m heads and dimensionality $O(\log n)$ that uses the Unified Attention Based Computer framework in Section 8.5.2 to implement a calculator which can perform addition, subtraction, multiplication, and computing the inverse, square root and percentage. For computing the inverse and square root, the operand needs to be in the range $[-e^{O(m)}, -\tilde{\Omega}(\frac{1}{\sqrt{m}})] \cup [\tilde{\Omega}(\frac{1}{\sqrt{m}}), e^{O(m)}]$ and $[0, O(m^2)]$ respectively, and the returned output is correct up to an error of $O(1/\sqrt{m})$ and $O(1/m)$ respectively. Here, n is the number of operations to be performed.*

Remark 8.18. *In the proof of this theorem, we use Theorem 8.13 to approximate the square root and the inversion function. That lemma provides error guarantees in terms of the number of heads m . We prove Theorem D.4 in the appendix which provides equivalent error guarantees, but where the error decreases with the dimension d of the transformer. Depending on the design choices of the*

transformer, either of the results can be used, and the calculator's error guarantee will also change accordingly.

We show how one can implement a calculator in our FLEQ framework in Algorithm 8.3.

Algorithm 8.3 A sample program for executing a basic calculator functionality. This algorithm performs $\frac{\sqrt{1/((a+b)-c) \cdot d}}{100}$

Require: $\text{mem}[p] = a, \text{mem}[q] = b, \text{mem}[r] = c, \text{mem}[s] = d.$	$\triangleright$ The location of the inputs.
1: $\text{mem}[t] = f_{\text{add}}(\text{mem}[p], \text{mem}[q])$	$\triangleright \text{mem}[t] = a + b.$
2: $\text{mem}[t] = f_{\text{sub}}(\text{mem}[t], \text{mem}[r])$	$\triangleright \text{mem}[t] = (a + b) - c.$
3: $\text{mem}[t] = f_{\text{mul}}(\text{mem}[t], \text{mem}[s])$	$\triangleright \text{mem}[t] = ((a + b) - c) * d.$
4: $\text{mem}[t] = f_{\text{inv}}(\text{mem}[t])$	$\triangleright \text{mem}[t] = 1/((a + b) - c) * d.$
5: $\text{mem}[t] = f_{\text{sqrt}}(\text{mem}[t])$	$\triangleright \text{mem}[t] = \sqrt{1/((a + b) - c) * d}.$
6: $\text{mem}[t] = f_{\text{perc}}(\text{mem}[t])$	$\triangleright \text{mem}[t] = \frac{\sqrt{1/((a+b)-c)*d}}{100}.$

Looking at the algorithm, it is clear that for proving the theorem above, it is sufficient to implement the 6 functions (addition, subtraction, multiplication, inversion, square root and percentage) using the transformer-based function blocks defined in Theorem 8.6. We start with two lemmas, which can be proved by constructing transformers that add and subtract in a similar way to the OISC transformer constructed in Section 8.5.1.

Lemma 8.19 (addition). *There exists a transformer-based function block with 3 layers, 1 head and dimensionality $O(1)$ which can implement $f(a, b) = a + b$.*

Proof. Consider the input in the form of Eq. (8.10)

$$\mathbf{X} = \left[\begin{array}{cc|cc|cc} a & \mathbf{0} & b & \mathbf{0} & 0 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{p}_{2d+1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{p}_{2:d} & \mathbf{p}_{d+1} & \mathbf{p}_{d+2:2d} & \mathbf{p}_{2d+1} & \mathbf{p}_{2d+2:3d} \\ 1 & \mathbf{0} & 0 & \mathbf{0} & 0 & \mathbf{0} \end{array} \right] \quad (8.11)$$

We can perform the following transformation

$$\left[\begin{array}{cc|cc|cc} a & \mathbf{0} & b & \mathbf{0} & 0 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{array} \right] \rightarrow \left[\begin{array}{cc|cc|cc} a & \mathbf{0} & b & \mathbf{0} & 0 & \mathbf{0} \\ a & \mathbf{0} & b & \mathbf{0} & 0 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{array} \right] \quad (8.12)$$

$$\rightarrow \left[\begin{array}{cc|cc|cc} a & \mathbf{0} & 0 & \mathbf{0} & 0 & \mathbf{0} \\ 0 & \mathbf{0} & b & \mathbf{0} & 0 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{array} \right] \quad (8.13)$$

$$\rightarrow \left[\begin{array}{cc|cc|cc} a+b & \mathbf{0} & 0 & \mathbf{0} & 0 & \mathbf{0} \\ 0 & \mathbf{0} & 0 & \mathbf{0} & 0 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{array} \right] \quad (8.14)$$

$$\rightarrow \left[\begin{array}{cc|cc|cc} a+b & \mathbf{0} & 0 & \mathbf{0} & a+b & \mathbf{0} \\ 0 & \mathbf{0} & b & \mathbf{0} & 0 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{array} \right] \quad (8.15)$$

The first and second step are implemented with one feed-forward layer each. The third step with the Section 8.4.2. We have ignored the last three rows since we don't change them and we only use them for the last step. $\square$

Lemma 8.20 (subtraction). *There exists a transformer-based function block with 3 layers, 1 head and dimensionality $O(1)$ which can implement $f(a, b) = a - b$.*

This lemma can be proved in the exact same way as the previous one. In addition, we can use the theory presented in Theorem 8.16 to get the following corollaries:

Corollary 8.21 (multiplication). *There exists a transformer-based function block with 2 layers, 1 head and dimensionality $O(d)$ which can implement $f(a, b) = ab$.*

Corollary 8.22 (percentage). *There exists a transformer-based function block with 2 layers, 1 head and dimensionality $O(1)$ which can implement $f(a) = a/100 = a * 0.01$.*

To implement inversion function, we introduce the following lemma.

Lemma 8.23. *Given $\epsilon, \delta \in [0, 1]$, and $C \geq 1$ there exists a function f of the form $f(x) = \sum_{i=1}^m c_i \sigma(w_i x + b_i)$, where σ is the sigmoid function, such that*

$$\forall x \in [\delta, C], \left| f(x) - \frac{1}{x} \right| \leq \epsilon,$$

as long as $d = \Omega\left(\frac{\log(1/(\epsilon\delta))}{\epsilon\delta} + \log C\right)$.

We can use this lemma along with the result presented in Theorem 8.13 to get the following corollary:

Corollary 8.24 (inversion). *There exists a transformer-based function block with 3 layers and m heads which can implement $f(a) = \frac{1}{a}$ up to error $\tilde{O}(\frac{1}{\sqrt{m}})$ for all $a \in [\tilde{\Omega}(\frac{1}{\sqrt{m}}), \tilde{O}(e^m)]$.*

Note that using Theorem 8.21 (multiplication) and Theorem 8.24 (inversion), the operation of division can be implemented as well. Next, we move on to showing the way of implementing square root.

Lemma 8.25. *Given $\epsilon \in [0, 1]$, and $C \geq 1$ there exists a function f of the form $f(x) = \sum_{i=1}^m c_i \sigma(w_i x + b_i)$, where σ is the sigmoid function such that*

$$\forall x \in [0, C], \left| f(x) - \sqrt{x} \right| \leq \epsilon,$$

as long as $m = \Omega\left(\frac{\sqrt{C}}{\epsilon}\right)$.

We can use this lemma along with the result presented in Theorem 8.13 to get the following corollary:

Corollary 8.26 (sqrt). *There exists a transformer-based function block with 3 layers and m heads which can implement $f(a) = \sqrt{a}$ up to error $O(1/m)$ for all $a \in [0, O(m^2)]$.*

The functions $f : x \rightarrow \frac{1}{x}$ (inversion) and $f : x \rightarrow \sqrt{x}$ (square root) since they can be approximated by sums of sigmoids, they can directly be encoded in the standard transformer-based function block form through Theorem 8.13.

What other functions can our calculator implement? We have included some of the most commonly used operations in calculators in our construction, but it can be extended to include a wider variety of operations such as algebraic and trigonometric functions. When implementing these functions within our transformer architecture, there are typically two choices that can be made. One option is to approximate the target function $f(x)$ using sigmoids. Another option is to use an iterative numerical algorithm where the next output y is calculated based on the previous output y and the goal is to minimize the difference between the calculated output and the target function $f(x)$. This algorithm takes the form $y_{k+1} = g(y_k)$, where g is typically an algebraic function. The desired accuracy is achieved when the difference between the calculated output and target function is less than or equal to a certain tolerance ϵ .

8.8 Linear Algebra

In Section 8.6, we demonstrated the implementation of matrix transpose and matrix multiplication as transformer-based function blocks. Utilizing these implementations, we proceed to execute two iterative algorithms for determining the inverse of a matrix through the Newton-Raphson Method and identifying the eigenvector corresponding to the maximum eigenvalue through the Power Iteration method.

Linear algebra using Transformers In the study conducted by Charton (2021), the author implemented some standard matrix method operations using a transformer-based architecture. Four distinct encoding schemes were proposed and applied to nine different

operations, ranging from matrix multiplication to eigenvalue decomposition. We find that the size of the networks in Charton (2021) is comparable to that of ours.

As an example we compare the required network size of ours and Charton (2021), for the task of transposing a matrix of size 30×30 : our construction uses a transformer with 1 layer, 1 head and width of 168, while the transformer in Charton (2021) has 1 layer, 8 heads and width of 256. Notice that the number of layers, heads and width reported above may seem different with Theorem 8.15; however, in the proof of Theorem 8.15 we first vectorize the matrix (1 layer), then we implement the fixed permutation using Theorem 8.4 (1 layer) and finally we use another 2 layers to bring back the matrix in its original representation. If the matrix is given to us, as in Charton (2021), in its transposed form then we only need one layer and the two sets of encodings to perform the fixed permutation. Since the maximum size of the matrix is 30×30 , the sequence length is $n = 30^2$ and thus the size of each of the encodings will be 10, leading to an input with width $2 \cdot 10 + 1 = 21$. This will lead to a total width of 168, due to the ReLU layer in Theorem D.1, for adding two binary vectors, having a width eight times the input's width.

We intend to further investigate our constructions, by implementing them and evaluating the errors involved as a function of the constants used in the proof of Theorem 8.16 and the temperature in Theorem 8.3, in future work.

Matrix Inversion. We can use the Unified Attention Based Computer to write a program for Matrix Inversion using the functions for matrix multiplications and a function for subtraction. We do so by implementing Newton's algorithm for matrix inversion using our unified framework. The pseudo code for the algorithm is as follows:

Algorithm 8.4 Pseudocode for running Newton's algorithm for Matrix inversion for T iterations.

```

1:  $\mathbf{X}_{-T} = \epsilon \mathbf{A}$ 
2: for  $i = -T, \dots, 0$  do
3:    $\mathbf{X}_{i+1} = \mathbf{X}_i(2\mathbf{I} - \mathbf{A}\mathbf{X}_i)$ 
4: end for
```

Lemma 8.27. Consider a matrix $\mathbf{A} \in \mathbb{R}^{d \times d}$, then for any $\epsilon > 0$ there exists a transformer with 13 layers, 1 head and dimensionality $r = O(d)$ that emulates Algorithm 8.4 with output $\mathbf{X}_1^{(transf)}$ that satisfies $\|\mathbf{X}_1^{(transf)} - \mathbf{X}_1\| \leq \epsilon$.

Proof. The proof of this lemma is the code using the FLEQ instruction provided below (Algorithm 8.5). Let f_{mul} , f_{sub} and f_{transp} be the functions that implement multiplication, subtraction and transpose respectively. Then, the following code runs Newton's algorithm for matrix inversion.

Algorithm 8.5 Program to compute the approximate inverse using our Unified Attention Based Computer

Require: $\text{mem}[a] = \mathbf{A}$. ▷ This is the location of the input.
Require: $\text{mem}[p] = 2\mathbf{I}$, $\text{mem}[x] = \epsilon\mathbf{I}$, $\text{mem}[y] = \mathbf{0}$, $\text{mem}[q] = -1$. ▷ Constants.
Require: $\text{mem}[t] = -T$. ▷ Iteration counter, i initialized as $i := -T$.

1: $\text{mem}[x] = f_{\text{mul}}(\text{mem}[x], \text{mem}[a])$. ▷ Initializes the result, $\mathbf{X}_{-T} := \epsilon\mathbf{A}$.
2: $\text{mem}[a] = f_{\text{transp}}(\text{mem}[a], \text{mem}[y])$ ▷ Transpose $\mathbf{A}$.
3: $\text{mem}[y] = f_{\text{mul}}(\text{mem}[a], \text{mem}[x])$. ▷ First sub-step of Newton's algorithm, $\mathbf{Y} := \mathbf{A}\mathbf{X}_i$
4: $\text{mem}[y] = f_{\text{sub}}(\text{mem}[p], \text{mem}[y])$. ▷ Second sub-step of Newton's algorithm, $\mathbf{Y} := 2\mathbf{I} - \mathbf{Y}$
5: $\text{mem}[y] = f_{\text{transp}}(\text{mem}[y], \text{mem}[q])$. ▷ Transpose of $\mathbf{Y}$.
6: $\text{mem}[x] = f_{\text{mul}}(\text{mem}[x], \text{mem}[y])$. ▷ Updating the result, $\mathbf{X}_{i+1} := \mathbf{X}_i \mathbf{Y}$
7: $\text{mem}[t] = f_{\text{sub}}(\text{mem}[t], \text{mem}[q])$. ▷ Increment counter, $i := i + 1$.
8: if $\text{mem}[t] \leq 0$ goto instruction 3. ▷ Keep looping back as long as $i \leq 0$.
9: EOF. ▷ End of File command.

Power Iteration. The Power Iteration algorithm (Algorithm 8.6) is used for finding the dominant eigenvalue, the one that has the maximum absolute value, and corresponding eigenvector of a diagonalizable matrix. The algorithm starts with an initial approximation of the eigenvector and converges linearly to the eigenvector associated with the dominant eigenvalue; below we provide the pseudocode.

Algorithm 8.6 Power Iteration

Input: $\mathbf{A}, T$
1: Initialize $b_0 = 1$
2: **for** $k = 0, \dots, T - 1$ **do**
3: $\mathbf{b}_{k+1} = \mathbf{A}\mathbf{b}_k$
4: **end for**
5: $\mathbf{b} = \frac{\mathbf{b}_T}{\|\mathbf{b}_T\|}$

The last step in the algorithm above needs a normalization by the norm of $\mathbf{b}_T$. While we can compute $\|\mathbf{b}_T\|^2$ easily and precisely using the matrix multiplication function block (since $\|\mathbf{b}_T\|^2 = \mathbf{b}_T^\top \mathbf{b}_T$), computing the norm and taking its inverse using the function block from Section 8.7 would induce error. Hence, we use the following Newton's algorithm that converges quadratically.

Algorithm 8.7 Newton's algorithm to compute inverse square root: $1/\sqrt{S}$

Input: S

- 1: Initialize $x_0 = 1$
 - 2: **for** $k = 0, \dots, T$ **do**
 - 3: $x_{k+1} = x_k \left(\frac{3}{2} - \frac{S}{2} x_k^2 \right)$
 - 4: **end for**
-

Lemma 8.28. *Consider a matrix $\mathbf{A} \in \mathbb{R}^{d \times d}$, then for any $\epsilon > 0$ there exists a transformer with 13 layers, 1 head and dimensionality $r = O(d)$ that emulates Algorithm 8.6 for $T = O(\log 1/\epsilon)$ iterations with output $\mathbf{b}_{T+1}^{(transf)}$ that satisfies $\|\mathbf{b}_{T+1}^{(transf)} - \mathbf{b}_{T+1}\| \leq \epsilon$.*

Proof. The proof consists of translating each step of the pseudocode for Algorithm 8.6 and Algorithm 8.7 to commands of our unified framework.

Algorithm 8.8 Program to simulate Power Iteration using our Unified Attention Based Computer

Require: $\text{mem}[a] = \mathbf{A}, \text{mem}[b] = \mathbf{1}, \text{mem}[\text{inv\_norm}] = 1.$ $\triangleright$ Location of matrix and initialization.

Require: $\text{mem}[q] = 1, \text{mem}[p] = 0, \text{mem}[r] = 0.5, \text{mem}[s] = 1.5$ $\triangleright$ Constants.

Require: $\text{mem}[t_1] = \text{mem}[t_2] = -T + 1,$

- 1: $\text{mem}[a] = f_{\text{transp}}(\text{mem}[a], \text{mem}[p]).$ $\triangleright$ Transpose of $\mathbf{A}.$
- 2: $\text{mem}[b] = f_{\text{mul}}(\text{mem}[a], \text{mem}[b]).$ $\triangleright$ Inner product: $\mathbf{A}\mathbf{b}_k.$
- 3: $\text{mem}[t] = f_{\text{add}}(\text{mem}[t_1], \text{mem}[q]).$ $\triangleright$ Increment counter, $i := i + 1.$
- 4: if $\text{mem}[t_1] \leq 0$ goto instruction 2. $\triangleright$ Keep looping back as long as $i \leq 0.$
- 5: $\text{mem}[\text{norm\_square}] = f_{\text{mul}}(\text{mem}[b], \text{mem}[b]).$ $\triangleright$ Calculate $\|\mathbf{b}_T\|^2.$
- Code for Algorithm 8.7 begins.
- 6: $\text{mem}[y] = f_{\text{mul}}(\text{mem}[\text{inv\_norm}], \text{mem}[\text{inv\_norm}]).$ $\triangleright$ Calculate $x_k^2.$
- 7: $\text{mem}[y] = f_{\text{mul}}(\text{mem}[\text{norm\_square}], \text{mem}[y]).$ $\triangleright$ Calculate $Sx_k^2.$
- 8: $\text{mem}[y] = f_{\text{mul}}(\text{mem}[r], \text{mem}[y]).$ $\triangleright$ Calculate $Sx_k^2/2.$
- 9: $\text{mem}[y] = f_{\text{sub}}(\text{mem}[s], \text{mem}[y]).$ $\triangleright$ Calculate $(3 - Sx_k^2)/2.$
- 10: $\text{mem}[\text{inv\_norm}] = f_{\text{mul}}(\text{mem}[\text{inv\_norm}], \text{mem}[y]).$ $\triangleright$ Update $x_{k+1} := x_k(3 - Sx_k^2)/2.$
- 11: $\text{mem}[t_2] = f_{\text{add}}(\text{mem}[t_2], \text{mem}[q]).$ $\triangleright$ Increment counter, $j := j + 1.$
- 12: if $\text{mem}[t_2] \leq 0$ goto instruction 6. $\triangleright$ Keep looping back as long as $j \leq 0.$
- Code for Algorithm 8.7 ends.
- 13: $\text{mem}[b] = f_{\text{mul}}(\text{mem}[b], \text{mem}[\text{inv\_norm}]).$ $\triangleright \mathbf{b} := \mathbf{b}_T / \|\mathbf{b}_T\|.$
- 14: EOF. $\triangleright$ End of File command.

What other numerical linear algebra algorithms can transformers implement? The algorithms presented above serve as proof of concept for the potential to build small linear algebra libraries using our transformer construction. As demonstrated, the size of the looped transformer is constant regardless of the depth. To implement iterative numerical algorithms, additional functions can be incorporated into our architecture. For instance, QR decomposition, Gauss-Seidel, Arnoldi iteration, or Lanczos algorithm can be implemented. While we have not included detailed code for these specific algorithms, the above examples should provide sufficient insight on how to do so.

8.9 Emulating Learning Algorithms at Inference Time

In this section we demonstrate the ability of our unified template to emulate Stochastic Gradient Descent (SGD). We begin by examining the case of linear models, before extending our results to the implementation of the backpropagation algorithm for two layer neural

networks. Utilizing this as a “function” which we call at each step, we demonstrate the application of SGD in updating the implicit weights of a model.

Our work demonstrates that looped transformers can effectively perform in-context learning for a wide range of models and achieve high levels of accuracy, given access to a sufficient number of inference calls/loops. Previous research, such as Akyürek et al. (2022) and Garg et al. (2022), has limited in-context learning to a single inference call of a transformer model deeper than ours, which restricts the types of models that can be learned and the level of accuracy that can be achieved. To implement complex iterative programs like SGD, either a looped structure transformer or one that grows in size with the program’s depth is required, unless widely believed complexity conjectures are falsified. Additionally, this is the first work to show that transformers can implement SGD on more general loss functions and models beyond linear regression.

Stochastic Gradient Descent in linear models. In Algorithm 8.9 we provide the program for running SGD in linear models; that is we perform updates of the form: $\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \sum_{i=1}^{\mathcal{D}} (\mathbf{w}^\top \mathbf{x}_i - y_i) \mathbf{x}_i$, where $\mathbf{w}$ is the weight vector, $(\mathbf{x}_i, y_i)$ is the feature-label pair of the i -th data point, and η is the step-size. The program iterates through the $\mathcal{D}$ data points that the user gives and cycles back to the first point after one pass is completed. The step-size is given as input by the user.

Lemma 8.29. *Let $\epsilon > 0$, there exists a transformer with 13 layers, 1 head and dimensionality $O(\log(\mathcal{D}) + d)$ that uses the Unified Attention Based Computer framework in Section 8.5.2 to implement T iterations of SGD on a weight vector $\mathbf{w} \in \mathbb{R}^d$, over a set of $\mathcal{D}$ data points $(\mathbf{x}_i, y_i) \in \mathbb{R}^{d+1}$, $i = 1, \dots, \mathcal{D}$ with error up to ϵ . The step size is given as a parameter to the program.*

Remark 8.30. *The error is controlled by two parameters: the temperature λ and the constants used in the proof of Theorem 8.16. Implementing arbitrary loss functions f and thus updates of the form $\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \sum_{i=1}^{\mathcal{D}} f'(\mathbf{w}^\top \mathbf{x}_i - y_i) \mathbf{x}_i$ would introduce an extra error as a result of Barron’s theorem (Theorem 8.12) applied in Theorem 8.13. Specifically, we would need in general $\text{poly}(T\mathcal{D})$ heads,*

in order to ensure control over this approximation error. However, if the derivative $f'(x)$ of the loss function $f(x)$ is a sum of sigmoids, the number of heads will be equal to the number of sigmoids required, and there will be no error associated with this aspect of the construction.

Algorithm 8.9 Program to simulate SGD using our Unified Attention Based Computer

Require: $\text{mem}[w] = \mathbf{w}, \text{mem}[\eta] = \eta.$	▷ Location of the weight and step-size.
Require: $\text{mem}[x_0 + i - 1] = \mathbf{x}_i, i = 1, \dots, \mathcal{D}.$	▷ Location of the data points.
Require: $\text{mem}[y_0 + i - 1] = y_i, i = 1, \dots, \mathcal{D}.$	▷ Location of the labels.
Require: $\mathbf{p}_{x_*} = x_0.$	▷ $\mathbf{p}_{x_*}$ is a pointer to the first data.
Require: $\mathbf{p}_{y_*} = y_0.$	▷ $\mathbf{p}_{y_*}$ is a pointer to the first label.
Require: $\mathbf{p}_{\text{PC}} = \text{instr}_1.$	▷ Program Counter points to first instruction.
Require: $\text{mem}[q] = 1, \text{mem}[p] = 0, \text{mem}[z] = n.$	▷ Constants.
Require: $\text{mem}[j] = -\mathcal{D}.$	▷ Within epoch iteration counter initialized to $-n.$
Require: $\text{mem}[k] = -T.$	▷ Epoch counter initialized to $-T.$
1: (instr ₁) $\text{mem}[\text{temp}] = f_{\text{mul}}(\text{mem}[\mathbf{p}_{x_*}], \text{mem}[w]).$	▷ Inner product: $\mathbf{w}^\top \mathbf{x}_i.$
2: (instr ₂) $\text{mem}[\text{temp}] = f_{\text{sub}}(\text{mem}[\text{temp}], \text{mem}[\mathbf{p}_{y_*}]).$	▷ Subtract the label: $\mathbf{w}^\top \mathbf{x}_i - y_i.$
3: (instr ₃) $\text{mem}[\text{temp}] = f_{\text{mul}}(\text{mem}[\mathbf{p}_{x_*}], \text{mem}[\text{temp}]).$	▷ Multiply with the data point $\mathbf{x}_i.$
4: $\text{mem}[\text{temp}] = f_{\text{mul}}(\text{mem}[\text{temp}], \text{mem}[\eta]).$	▷ Multiply with the step-size.
5: $\text{mem}[w] = f_{\text{sub}}(\text{mem}[w], \text{mem}[\text{temp}]).$	▷ Subtract from $\mathbf{w}$ one gradient step.
6: $\text{mem}[\text{instr}_1] = f_{\text{incr\_pointer}}(\text{mem}[\text{instr}_1]).$	▷ Increment pointer.
7: $\text{mem}[\text{instr}_2] = f_{\text{incr\_pointer}}(\text{mem}[\text{instr}_2]).$	▷ Increment pointer.
8: $\text{mem}[\text{instr}_3] = f_{\text{incr\_pointer}}(\text{mem}[\text{instr}_3]).$	▷ Increment pointer.
9: $\text{mem}[j] = f_{\text{add}}(\text{mem}[j], \text{mem}[q]).$	▷ Increment within epoch iteration counter by 1.
10: if $\text{mem}[j] \leq 0$ goto 1.	▷ Cycle back to the first data point.
11: $\text{mem}[j] = -\mathcal{D}.$	▷ Reset counter.
12: $\text{mem}[\text{instr}_1] = f_{\text{reset\_pointer}}(\text{mem}[\text{instr}_1], x_0).$	▷ Reset pointer.
13: $\text{mem}[\text{instr}_2] = f_{\text{reset\_pointer}}(\text{mem}[\text{instr}_2], y_0).$	▷ Reset pointer.
14: $\text{mem}[\text{instr}_3] = f_{\text{reset\_pointer}}(\text{mem}[\text{instr}_3], x_0).$	▷ Reset pointer.
15: $\text{mem}[k] = f_{\text{add}}(\text{mem}[k], \text{mem}[q]).$	▷ Increment epoch counter by 1.
16: if $\text{mem}[k] \leq 0$ goto 1.	▷ Cycle back to the first data point.
17: EOF.	▷ End of File command.

The following will detail the essential procedures for implementing the Stochastic Gradient Descent algorithm. We employ three pointers, namely $\mathbf{p}_{\text{PC}}$, $\mathbf{p}_{x_*}$ and $\mathbf{p}_{y_*}$ in our algorithm. The first one, referred to as program counter, is used to iterate through the commands; after one pass over all data points is completed, the program counter is reset to the first instruction (line 16), until T full passes have been completed. The second and third ones, referred to as data and label pointer respectively, iterate through the features and labels one by one. The increment of the pointer $\mathbf{p}_{x_*}$ needs to occur in both instructions

1 and 3, as to in the next iteration they have been updated from $\text{instr}_i(\mathbf{p}_{x_*}, w, temp) \rightarrow \text{instr}_i(\mathbf{p}_{x_*} + 1, w, temp)$, $i = 1, 3$. The same holds for the pointer $\mathbf{p}_{y_*}$ in line 7. Finally, we reset the two pointers in lines 13,14 to cycle back in the first feature and label.

To enhance understanding, we note that lines 6-8 modify the instructions themselves; instead of doing this we could have $\mathcal{D}$ copies of the lines 1-3, each one with parameters pointers of a different (feature,label) pair. In that case the number of *instructions* would have been $7\mathcal{D}$.

Notice that the functions $f_{\text{incr\_pointer}}$ and $f_{\text{reset\_pointer}}$ can be directly implemented using Theorem D.1.

Backpropagation and SGD. We will now generalize the result of Theorem 8.29 to two layer neural networks with non-linear activation functions; we demonstrate in Algorithm 8.12 how this can be achieved if the activation function is the sigmoid function.

Closest to this section is the work of Akyürek et al. (2022), where the authors prove that constant number of layers is needed to perform one step SGD in linear models, using decoder only transformer architecture.

Algorithm 8.10 Backpropagation

- Loss function: $J(x) = \frac{1}{2}x^2$
Input: $\mathbf{W}_1 \in \mathbb{R}^{m \times d}$, $\mathbf{b}_1 \in \mathbb{R}^m$, $\mathbf{W}_2 \in \mathbb{R}^{m \times 1}$, $\mathbf{b}_2 \in \mathbb{R}$ $\mathbf{x} \in \mathbb{R}^d$, $y \in \mathbb{R}$
- 1: Compute $\mathbf{z} = \mathbf{W}_1 \mathbf{x} + \mathbf{b}_1$.
 - 2: Compute $\mathbf{a} = \sigma(\mathbf{z})$.
 - 3: Compute $o = \mathbf{W}_2 \mathbf{a} + \mathbf{b}_2$.
 - 4: Compute $\delta_2 = (o - y)$.
 - 5: Compute $\delta_1 = \sigma'(\mathbf{z}) \odot \mathbf{W}_2(o - y)$.
 - 6: Compute $\frac{\partial J}{\partial \mathbf{W}_2} = \delta_2 \mathbf{a}^\top$.
 - 7: Compute $\frac{\partial J}{\partial \mathbf{b}_2} = \delta_2$.
 - 8: Compute $\frac{\partial J}{\partial \mathbf{W}_1} = \delta_1 \mathbf{x}^\top$.
 - 9: Compute $\frac{\partial J}{\partial \mathbf{b}_1} = \delta_1$.
-

Lemma 8.31. *Let $\epsilon > 0$, there exists a transformer with 13 layers, 1 head and dimensionality $O(\log(\mathcal{D}) + d)$ that uses the Unified Attention Based Computer framework in Section 8.5.2 to*

implement T iterations of SGD on a two layer neural network, over a set of $\mathcal{D}$ data points $(\mathbf{x}_i, y_i) \in \mathbb{R}^{d+1}$, $i = 1, \dots, \mathcal{D}$ with error up to ϵ . The step size is given as a parameter to the program.

Remark 8.32. The program we provide in Algorithm 8.11 is implemented as an independent function, which we call multiple times. Specifically, in line 1 of Algorithm 8.12 we call the algorithm for backpropagation at each iteration with a different data point. In terms of our construction, this translates to different instructions which will be in total $O(\mathcal{D})$, each one with parameters pointers to a different data point. However, as in Algorithm 8.9 the utilization of a pointer that changes the instructions themselves, would result in a program of constant length; we did not do this in order to contain the total length of the program.

Remark 8.33. If we want to account for different activation functions we can use Theorem 8.13 to express the activation function and its derivative as sums of sigmoids. The number of heads would need to be in that case $\text{poly}(T\mathcal{D})$ to ensure control over the error induced by the approximation.

Algorithm 8.11 Program to simulate Backpropagation for two layer Neural Networks

Input: $\mathbf{p}_{w_1}, \mathbf{p}_{w_2}, \mathbf{p}_{b_1}, \mathbf{p}_{b_2}$	▷ Pointers to weights and biases.
Input: $\mathbf{p}_x, \mathbf{p}_y$	▷ Pointer to data point and label.
Input: η .	▷ Pointer to step size.
Require: $\text{mem}[q] = 1, \text{mem}[p] = 0, \text{mem}[r] = -1, \text{mem}[m] = m$.	▷ Constants.
Require: $\text{mem}[k] = 1$.	▷ Iteration counter, $k := 1$.
Require: $\mathbf{p}_z = z_T^1$.	▷ Pointer for z .
Require: $\mathbf{p}_\delta = \delta_{1,T}^1$.	▷ Pointer for δ_1 .
1: (instr ₁) $\text{mem}[\text{temp}] = f_{\text{trans}}(\text{mem}[\mathbf{p}_{w_1}], \text{mem}[p])$.	▷ Create $\mathbf{W}_1^T$.
2: $\text{mem}[z] = f_{\text{mul}}(\text{mem}[\text{temp}], \text{mem}[\mathbf{p}_x])$.	▷ Multiply: $\mathbf{W}_1 \mathbf{x}$.
3: $\text{mem}[z] = f_{\text{add}}(\text{mem}[z], \text{mem}[\mathbf{p}_{b_1}])$.	▷ Add the bias: Compute z .
4: $\text{mem}[a] = f_{\text{sigmoids}}(\text{mem}[z], \text{mem}[q])$.	▷ Compute $a = \sigma(z)$.
5: $\text{mem}[\text{temp}] = f_{\text{trans}}(\text{mem}[\mathbf{p}_{w_2}], \text{mem}[p])$.	▷ Create $\mathbf{W}_2^T$.
6: $\text{mem}[o] = f_{\text{mul}}(\text{mem}[\text{temp}], \text{mem}[a])$.	▷ Multiply: $\mathbf{W}_2 a$.
7: $\text{mem}[o] = f_{\text{add}}(\text{mem}[o], \text{mem}[\mathbf{p}_{b_2}])$.	▷ Add bias: Compute o .
8: $\text{mem}[\delta_2] = f_{\text{sub}}(\text{mem}[o], \text{mem}[\mathbf{p}_y])$.	▷ Compute δ_2 .
9: $\text{mem}[\delta_1] = f_{\text{mul}}(\text{mem}[\mathbf{p}_{w_2}], \text{mem}[\delta_2])$.	▷ Multiply $\mathbf{W}_2 \delta_2$.
10: $\text{mem}[\text{flag}] = f_{\text{sub}}(\text{mem}[k], \text{mem}[m])$.	▷ Create $k - m$.
11: $\text{mem}[\mathbf{p}_z] = f_{\text{trans}}(\text{mem}[z], \text{mem}[p])$.	▷ Store z to consecutive memory cells.
12: $\text{mem}[\mathbf{p}_\delta] = f_{\text{trans}}(\text{mem}[\delta_1], \text{mem}[p])$.	▷ Store δ_1 to consecutive memory cells.
13: if $\text{mem}[\text{flag}] \leq 0$ goto 20.	▷ If we iterated all the elements goto next command.
14: (instr ₁₄) $\text{mem}[\text{temp}'] = f_{\text{sigmoids}}(\text{mem}[p], \text{mem}[\mathbf{p}_z])$.	▷ Create $\sigma(z_i)$.
15: $\text{mem}[\text{temp}'] = f_{\text{sub}}(\text{mem}[q], \text{mem}[\text{temp}'])$.	▷ Create $1 - \sigma(z_i)$.
16: $\text{mem}[\text{temp}'] = f_{\text{mul}}(\text{mem}[\text{temp}'], \text{mem}[\text{temp}''])$.	▷ Create $\sigma'(z_i) = \sigma(z_i)(1 - \sigma(z_i))$.
17: (instr ₁₇) $\text{mem}[\mathbf{p}_\delta] = f_{\text{mul}}(\text{mem}[\text{temp}'], \text{mem}[\mathbf{p}_\delta])$.	▷ Create $\sigma'(z_i)(\mathbf{W}_2)_i(o - y)$.
18: $\text{mem}[\text{instr}_{14}] = f_{\text{incr\_pointer}}(\text{mem}[\text{instr}_{14}])$.	▷ Point to next element of z .
19: $\text{mem}[\text{instr}_{17}] = f_{\text{incr\_pointer}}(\text{mem}[\text{instr}_{17}])$.	▷ Point to next element of δ_1 .
20: $\text{mem}[k] = f_{\text{add}}(\text{mem}[k], \text{mem}[q])$.	▷ Increment counter, $k := k + 1$.
21: If $\text{mem}[p] \leq 0$ goto 13.	▷ Loop back.
22: $\text{mem}[\text{instr}_1] = f_{\text{reset\_pointer}}(\text{mem}[\text{instr}_{14}], z_T^1)$.	▷ Reset pointer.
23: $\text{mem}[\text{instr}_{15}] = f_{\text{reset\_pointer}}(\text{mem}[\text{instr}_{15}], \delta_{1,T}^1)$.	▷ Reset pointer.
24: $\text{mem}[\text{grad}_{W_2}] = f_{\text{mul}}(\text{mem}[\delta_2], \text{mem}[a])$.	▷ Create $\frac{\partial J}{\partial \mathbf{W}_2}$.
25: $\text{mem}[\text{grad}_{b_2}] = f_{\text{mul}}(\text{mem}[\delta_2], \text{mem}[q])$.	▷ Create $\frac{\partial J}{\partial \mathbf{b}_2}$.
26: $\text{mem}[\text{grad}_{W_1}] = f_{\text{mul}}(\text{mem}[\delta_1], \text{mem}[\mathbf{p}_x])$.	▷ Create $\frac{\partial J}{\partial \mathbf{W}_1}$.
27: $\text{mem}[\text{grad}_{b_1}] = f_{\text{mul}}(\text{mem}[\delta_1], \text{mem}[q])$.	▷ Create $\frac{\partial J}{\partial \mathbf{b}_1}$.
28: $\text{mem}[\text{temp}] = f_{\text{mul}}(\text{mem}[\text{grad}_{W_2}], \text{mem}[\eta])$.	▷ Multiply with step-size.
29: $\text{mem}[\mathbf{p}_{w_2}] = f_{\text{sub}}(\text{mem}[\mathbf{p}_{w_2}], \text{mem}[\text{temp}])$.	▷ Update $\mathbf{W}_2$.
30: $\text{mem}[\text{temp}] = f_{\text{mul}}(\text{mem}[\text{grad}_{W_1}], \text{mem}[\eta])$.	▷ Multiply with step-size.
31: $\text{mem}[\mathbf{p}_{w_1}] = f_{\text{sub}}(\text{mem}[\mathbf{p}_{w_1}], \text{mem}[\text{temp}])$.	▷ Update $\mathbf{W}_1$.
32: $\text{mem}[\text{temp}] = f_{\text{mul}}(\text{mem}[\text{grad}_{b_2}], \text{mem}[\eta])$.	▷ Multiply with step-size.
33: $\text{mem}[\mathbf{p}_{b_2}] = f_{\text{sub}}(\text{mem}[\mathbf{p}_{b_2}], \text{mem}[\text{temp}])$.	▷ Update $\mathbf{b}_2$.
34: $\text{mem}[\text{temp}] = f_{\text{mul}}(\text{mem}[\text{grad}_{b_1}], \text{mem}[\eta])$.	▷ Multiply with step-size.
35: $\text{mem}[\mathbf{p}_{b_1}] = f_{\text{sub}}(\text{mem}[\mathbf{p}_{b_1}], \text{mem}[\text{temp}])$.	▷ Update $\mathbf{b}_1$.

Algorithm 8.12 Program to simulate SGD using our Unified Attention Based Computer

Require: $\text{mem}[w_1] = \mathbf{W}_1, \text{mem}[w_2] = \mathbf{W}_2.$ ▷ Location weights and biases.
Require: $\text{mem}[b_1] = \mathbf{b}_1, \text{mem}[b_2] = \mathbf{b}_2.$ ▷ Location of biases.
Require: $\text{mem}[x_0 + i - 1] = \mathbf{x}_i, i = 1, \dots, \mathcal{D}.$ ▷ Location of the data points.
Require: $\text{mem}[y_0 + i - 1] = y_i, i = 1, \dots, \mathcal{D}.$ ▷ Location of the labels.
Require: $\text{mem}[z] = e.$ ▷ Indicator for the choice of loss function
Require: $\mathbf{p}_{x_*} = x_0.$ ▷ $\mathbf{p}_{x_*}$ is a pointer to the first data.
Require: $\mathbf{p}_{y_*} = y_0.$ ▷ $\mathbf{p}_{y_*}$ is a pointer to the first label.
Require: $\mathbf{p}_{\text{PC}} = \text{instr}_1.$ ▷ Program Counter points to first instruction.
Require: $\text{mem}[q] = 1, \text{mem}[p] = 0, \text{mem}[z] = n.$ ▷ Constants.
Require: $\text{mem}[j] = -\mathcal{D}.$ ▷ Within epoch iteration counter initialized to $-n$.
Require: $\text{mem}[k] = -T.$ ▷ Epoch counter initialized to $-T$.

- 1: $\text{Backpropagation}(w_1, w_2, b_1, b_2, \mathbf{p}_{x_*}, \mathbf{p}_{y_*})$ ▷ Perform one step of SGD using Backpropagation
- 2: $\text{mem}[j] = f_{\text{add}}(\text{mem}[j], \text{mem}[q]).$ ▷ Increment within epoch iteration counter by 1.
- 3: $\mathbf{p}_{x_*} = f_{\text{incr\_pointer}}(\mathbf{p}_{x_*}).$ ▷ Show to next data point.
- 4: $\mathbf{p}_{y_*} = f_{\text{incr\_pointer}}(\mathbf{p}_{y_*}).$ ▷ Show to next label.
- 5: if $\text{mem}[j] \leq 0$ goto 1. ▷ Cycle back until all data points are iterated.
- 6: $\text{mem}[j] = -\mathcal{D}.$ ▷ Reset counter.
- 7: $\mathbf{p}_{x_*} = f_{\text{reset\_pointer}}(\mathbf{p}_{x_*}, x_0).$ ▷ Reset pointer.
- 8: $\mathbf{p}_{y_*} = f_{\text{reset\_pointer}}(\mathbf{p}_{y_*}, y_0).$ ▷ Reset pointer.
- 9: $\text{mem}[\text{instr}_3] = f_{\text{reset\_pointer}}(\text{mem}[\text{instr}_3], x_0).$ ▷ Reset pointer.
- 10: $\text{mem}[k] = f_{\text{add}}(\text{mem}[k], \text{mem}[q]).$ ▷ Increment epoch counter by 1.
- 11: if $\text{mem}[k] \leq 0$ goto 1. ▷ Cycle back to the first data point.
- 12: EOF. ▷ End of File command.

Generalizing to arbitrary depth. Our algorithm above is designed to emulate backpropagation on a neural network that contains only one hidden layer. However, it is important to note that this construction can be generalized to networks of arbitrary depth, with the caveat that the length of the code will scale with the number of layers in the network. This is because each line of code in our algorithm represents one cycle of the looped transformer, and the number of cycles required is directly proportional to the depth of the network. It's important to note that the number of cycles of the looped transformer will be equal to the depth of the network. So the cost of this algorithm is proportional to looping the transformer network as many times as the depth of the network. This means that as the network becomes deeper, the computational cost of training it using our algorithm will also increase.

8.10 Conclusion and Open Problems

In this chapter, we have shown that transformer networks can be used as universal computers by programming them with specific weights and placing them in a loop. We demonstrate that a constant number of encoder layers can emulate basic computing blocks, such as lexicographic operations, non-linear functions, function calls, program counters, and conditional branches. We construct a one-instruction set computer (OISC) and use it to map iterative algorithms to programs that can be executed by a transformer network. Our results include constant-depth transformers that emulate a basic calculator, a basic linear algebra library, and even a full backpropagation, in-context learning algorithm. Our findings reveal the potential of transformer networks as programmable compute units and offer insight into the mechanics of attention.

Our study sheds light on the versatility of the attention mechanism and how even a single loop can enable the creation of models that can mimic complex iterative algorithms and execute general programs. Our findings also reveal the ability of transformer models to effectively perform intricate mathematical and algorithmic tasks. It is possible that advanced transformer models like GPT-3 use similar internal subroutines when given in-context examples and instructions. In a sense, these models may have the ability to call upon a specific skill or algorithm, similar to a function call, when given contextual examples and instructions. The unique aspect of this is that the programming language of transformers is in natural language, rather than traditional code. This opens up the possibility of using natural language commands to control and program these models, further expanding their potential as programmable computers.

In conclusion, there are several open problems that warrant further exploration in the field of programmable computers using transformer networks. One of the most intriguing possibilities is the potential to fuse hardcoded models with larger pretrained transformers, in order to harness the strengths of both. Additionally, as our constructions currently do not take into account the language aspect of the input, it would be interesting to

investigate ways to tokenize input commands in order to map them to natural language. Another promising avenue for research is the potential for model distillation, in which larger networks could learn the skills performed by these looped transformers. Additionally, experimental validation through the creation of even smaller networks, trained on input-output pairs as well as internal representations, could provide further insight into the capabilities of these designs. Finally considering what architecture changes would make the above designs easier to implement and train, could lead to new insights in the field.

9 EXPRESSIVE POWER OF NORMALIZATION LAYERS

Feature normalization transforms such as Batch and Layer-Normalization have become indispensable ingredients of state-of-the-art deep neural networks. Recent studies on fine-tuning large pretrained models indicate that just tuning the parameters of these affine transforms can achieve high accuracy for downstream tasks. These findings open the questions about the expressive power of tuning the normalization layers of frozen networks. In this work, we take the first step towards this question and show that for random ReLU networks, fine-tuning only its normalization layers can reconstruct any target network that is $\mathcal{O}(\sqrt{\text{width}})$ times smaller. We show that this holds even for randomly sparsified networks, under sufficient overparameterization, in agreement with prior empirical work.

9.1 Introduction

Many modern machine learning techniques work by training or tuning only a small part of a pretrained network, rather than training all the weights from scratch. This is particularly useful in tasks like transfer learning (Yosinski et al., 2014; Donahue et al., 2014; Guo et al., 2020; Houlsby et al., 2019; Zaken et al., 2021), multitask learning (Mudrakarta et al., 2018; Clark et al., 2019), and few-shot learning (Lifchitz et al., 2019). Fine-tuning only a subset of the parameters of a large-scale model allows not only significantly faster training, but can sometimes lead to better accuracy than training from scratch (Zaken et al., 2021).

One particular way of model fine-tuning is to train only the Batch Normalization (Batch-Norm) or Layer Normalization (LayerNorm) parameters (Mudrakarta et al., 2018; Huang and Belongie, 2017). These normalization layers typically operate as affine transformations of each activation output, and as such one would expect that their expressive power is

This chapter presents our paper "The Expressive Power of Tuning Only the Norm Layers" (Giannou et al., 2023).

small, especially in comparison to tuning the weight matrices of a network. However, in an extensive experimental study, Frankle et al. (2020c) discovered that training these normalization parameters in isolation leads to predictive models with accuracy far above random guessing, even when all model weights are frozen at random values. The authors further showed that increasing the width/depth of these random networks allowed normalization layers training to reach significant accuracy across CIFAR-10 and ImageNet, and higher in comparison to training subsets of the network with similar number of parameters as that of normalization layers.

The above experimental studies indicate that training only the normalization layers of a network, seems—at least in practice—expressive enough to allow non-trivial accuracy for a variety of target tasks. In this work, we make a first step towards theoretically exploring the above phenomenon, and attempt to tackle the following open question:

What is the expressive power of tuning only the normalization layers of a neural network?

At first glance, it does not seem that training only the normalization layers has large expressive power. This is because, as we argue in Section 9.2, at its core, training the normalization layer parameters is equivalent to scaling the input of each neuron and adding a bias term. Let $\mathbf{W}$ be a $d \times d$ weight matrix of a given layer, $\mathbf{\Gamma}$ a diagonal matrix (meaning that has everywhere zero values except for the ones in the diagonal) that normalizes the activation outputs and β a d -dimensional vector that acts as a bias correction term. Then we have that the output of such a layer is equal to

$$f_{\mathbf{\Gamma},\beta}(\mathbf{x}) = \sigma(\mathbf{\Gamma}\mathbf{W}\mathbf{x} + \beta), \quad (9.1)$$

where σ is an element-wise activation and $\mathbf{x} \in \mathbb{R}^d$ is the input to that layer.

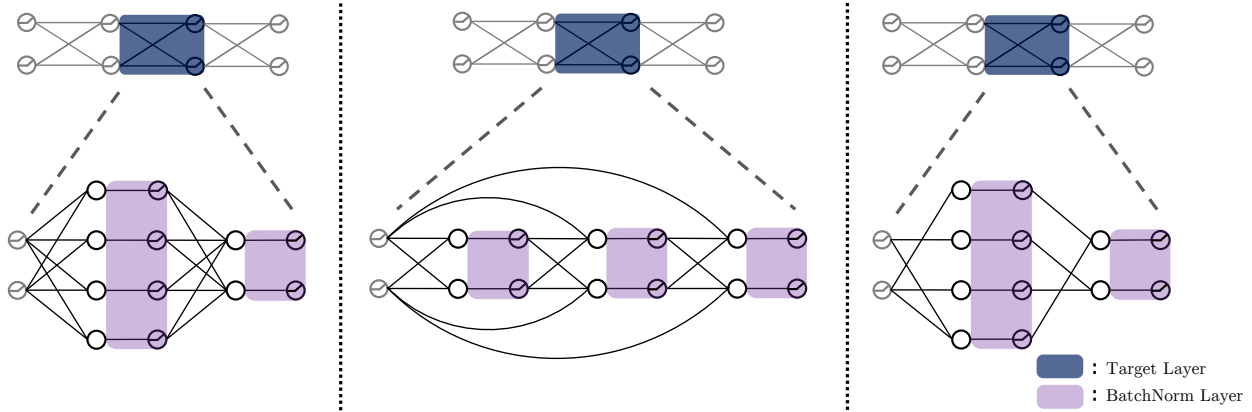


Figure 9.1: Each fully connected layer of a target network can be exactly recovered by either: (left) two polynomially wider layers, whose only normalization parameters are tunable or (middle) a polynomially deeper layer with the same width and skip connections, whose only normalization parameters are tunable. We also show that either of these two cases can also work when the random network has sparse weight matrices (right).

Tuning just these two sets of parameters, *i.e.*, Γ and β , seems way less expressive than training W , which contains a factor of $\mathcal{O}(d)$ more trainable parameters. An even more striking expressive disadvantage of the normalization layer parameters is that they do not linearly combine the individual coordinates of x . A normalization layer only scales and shifts a layer's input. How is it then possible to get high accuracy results by training these simple affine transformations of internal features?

Our contributions. In this paper, we theoretically investigate the expressive power of the normalization layers. In particular, we prove that any given neural network can be perfectly reconstructed by only tuning the normalization layers of a wider, or deeper random network that contains only a factor of $\tilde{\mathcal{O}}(\sqrt{d})$ more parameters (including both trainable and random).

Theorem (Informal). *Let g be any fully connected neural network with l layers and width d . Then, any randomly initialized fully connected network f with l' layers and width d' , with normalization layers can exactly recover the network g functionally, by tuning only the normalization layer parameters, as long as $d'l' \geq 2d^2l$, $d' \geq d$ and $l' \geq 2l$. Further, if f has sparse weight matrices, then*

the total number of parameters (trainable and random) only needs to be a factor of $\tilde{O}(\sqrt{d})$ larger than the target network.

Remark 9.1. Note that for the case when $d' = \mathcal{O}(d^2)$ and $l' = \mathcal{O}(l)$, the number of **trainable** parameters for both the networks are still of the same order, which is d^2l . This is because only the diagonal elements of the normalization layer matrix are trainable, which are of the order of $d' = d^2$. This distinction between the total number of parameters and trainable parameters is particularly important in our work's context because the total number of parameters in f would usually be of the order $\mathcal{O}(d^3l)$ (as we will see later), however the number of trainable parameters (that is, just the normalization layer parameters) would still be the same.

A sketch of our construction of random frozen network with tunable normalization layers that allows this result is shown in Fig. 9.1. The first construction in the figure reconstructs each layer of the target network using two layers of a wider, frozen random neural network with tunable normalization layer parameters. We then show that if skip connections are allowed, we can reconstruct the target network by a much narrower yet deeper frozen random network with tunable normalization layers. This indicates that adding the skip connections can potentially increase the expressive power of deep networks with normalization layers.

We provide the formal theorem statements for each of the three cases: wide, deep, and sparse reconstructions in Sections 9.4.1, 9.4.2, and 9.4.3 respectively. The proofs of the theorems rely crucially on the invertibility of Khatri-Rao products of random (possibly sparse) matrices. Due to the complex structure of the Khatri-Rao product, it introduces dependencies amongst the entries of the matrix and consequently, proving the invertibility becomes challenging; while in the case of reconstruction by a deep neural network, some more complicated arguments are required regarding the multiplication of random matrices. We provide proof sketches in the main text of the paper (Sections 9.4.1, 9.4.2 and 9.4.3), while the full proofs are deferred to the Appendix.

9.1.1 Related Work

Feature Normalization techniques are widely known to improve generalization performance and accelerate training in deep neural networks. The first normalization technique introduced was Batch Normalization by Ioffe and Szegedy (2015b), followed by Weight Normalization (Salimans and Kingma, 2016) and Layer Normalization (Ba et al., 2016). The initial method of Batch Normalization claimed to reduce the *internal covariate shift*, which Ioffe and Szegedy (2015b) define as the change in network parameters of the layers preceding to any given layer. The expectation is that by ensuring that the layer inputs are always zero mean and unit variance, training will be faster, similar to how input normalization typically helps training (LeCun et al., 2012; Wiesler and Ney, 2011).

In a subsequent work, Balduzzi et al. (2017) showed how BatchNorm leads to different neural activation patterns for different inputs; this might be connected to better generalization (Morcos et al., 2018). The claim concerning the internal covariate shift was later doubted by the experimental study of Santurkar et al. (2018), in which the authors show that adding noise with non-zero mean after Batch Normalization still leads to fast convergence. A potential explanation is that the loss landscape becomes smoother, a direction explored by Bjorck et al. (2018), who show that large step sizes lead to large gradient norms if Batch Normalization is not used.

For learning half-spaces using linear models with infinitely differentiable loss functions, Kohler et al. (2019) show that Batch Normalization leads to exponentially fast convergence. They show that it reparameterizes the loss such that the norm and direction component of weight vectors become decoupled, similar to weight normalization (Salimans and Kingma, 2016; Gitman and Ginsburg, 2017). Yang et al. (2019) use mean-field theory to show how residual connections help stabilize the training for networks with Batch Normalization. Luo et al. (2018) show that BatchNorm can be decomposed into factors that lead to both implicit and explicit regularizations.

Tuning only the Batch Normalization layers of a random network is related to learning

with random features. The works of Rahimi and Recht (2007, 2008a,b) focused on the power of random features, both theoretically and empirically in the context of kernel methods and later on training neural networks. Andoni et al. (2014) investigate linear combinations of random features for approximating bounded degree polynomials with complex initialization, through the lens of two layer ReLU networks. In the same context, Bach (2017), Ji et al. (2019) give upper bounds on the width for approximating Lipschitz functions. There are other multiple works that give upper bounds on the necessary width for approximating with combinations of random features, different classes of functions (Barron, 1993b; Klusowski and Barron, 2016; Sun et al., 2018; Hsu et al., 2021).

Another result considering random features concerns the overparameterized setting: *Stochastic Gradient Descent* tends to leave the first layer almost intact (Du et al., 2018). Thus, in the limit the first layer can be considered as randomly initialized (Yehudai and Shamir, 2019; Jacot et al., 2018). Building on this, Yehudai and Shamir (2019) showed that $\text{poly}(d)$ –overparameterized two layer ReLU neural network with random features can learn polynomials by training the second layer with SGD .

On the negative side, there is a line of research on impossibility results in the approximation power of two layer ReLU neural networks. Ghorbani et al. (2019) showed the limitation in the high dimensional setting for approximating high degree polynomials. Yehudai and Shamir (2019) showed that in order to approximate a single ReLU neuron with a two layer ReLU network exponential overparameterization is needed. However, Hsu et al. (2021) give matching upper and lower bounds on the number of neurons needed to approximate any L –Lipschitz function. Using a different approximation scheme they achieve polynomial in the dimension bounds for 1– Lipschitz functions.

9.2 Preliminaries

Notation. We use bold capital letters to denote matrices (e.g., $\mathbf{V}$, $\mathbf{W}$, $\mathbf{A}$); bold lowercase letters to denote vectors (e.g., $\mathbf{v}$, $\mathbf{w}$, $\mathbf{a}$); $\mathbb{R}^{n \times m}$ to denote the space of all $n \times m$ real matrices;

$\otimes, \odot$, and $\circ$ to denote the Kronecker, the Khatri-Rao and the Hadamard products respectively (cf. Definitions 9.3 and 9.4); and $[i]$ to denote the set of integers $\{1, \dots, i\}$.

When referring to randomly initialized neural networks, we imply that each weight matrix has independent, identically distributed elements, drawn from any arbitrary continuous and bounded distribution. We consider neural networks with ReLU activations, and denote the activation function by σ .

Definition 9.2 (Equivalence / Realization). *We say that two neural networks f and g , possibly with different architectures and/or weights are functionally equivalent on a given domain $\mathcal{D}$, if $\forall \mathbf{x} \in \mathcal{D}$, $f(\mathbf{x}) = g(\mathbf{x})$. We denote this by $f \equiv g$.*

This is also the same as saying that f and g have the *same realization* (Gribonval et al., 2022), or f realizes g and vice-versa. This work focuses on bounded domains, so $\mathcal{D} = \{\mathbf{x} | \mathbf{x} \in \mathbb{R}^d \wedge \|\mathbf{x}\| \leq 1\}$ unless stated otherwise.

Linear Algebra. For convenience, we provide the definitions of Khatri-Rao and the Hadamard products below:

Definition 9.3 (Khatri-Rao product). *The Khatri-Rao product of two matrices $\mathbf{A} \in \mathbb{R}^{n \times mn}$, $\mathbf{B} \in \mathbb{R}^{m \times mn}$ is defined as the matrix that contains column-wise Kronecker products. Formally,*

$$\mathbf{A} \odot \mathbf{B} = \begin{bmatrix} \mathbf{A}_1 \otimes \mathbf{B}_1 & \mathbf{A}_2 \otimes \mathbf{B}_2 & \dots & \mathbf{A}_{mn} \otimes \mathbf{B}_{mn} \end{bmatrix} \quad (9.2)$$

where $\mathbf{A}_i, \mathbf{B}_i$ denote the i -th column of $\mathbf{A}$ and $\mathbf{B}$ respectively.

Definition 9.4 (Hadamard Product). *Let $\mathbf{A} \in \mathbb{R}^{n \times m}$ and $\mathbf{B} \in \mathbb{R}^{n \times m}$; then their Hadamard product is defined as the element-wise product between the two matrices. Formally,*

$$(\mathbf{A} \circ \mathbf{B})_{i,j} = (\mathbf{A})_{i,j}(\mathbf{B})_{i,j}$$

Batch Normalization. Batch Normalization was introduced by Ioffe and Szegedy (2015b). However, over time two major variants have been developed: in the first one, the normalization with batch mean and variance is done before applying the affine transformation with the layer's weight parameters. Using the same notation from (9.1), we can write this layer as:

$$f_{\Gamma,\beta}(\mathbf{x}) = \sigma(\mathbf{W}^\top (\Gamma(\epsilon\mathbf{I} + \Sigma_{\text{pre}})^{-1}(\mathbf{x} - \boldsymbol{\mu}_{\text{pre}}) + \boldsymbol{\beta}) + \mathbf{b}),$$

where the mean $\boldsymbol{\mu}_{\text{pre}}$ and variance Σ_{pre} are those of the raw inputs to the layer $\mathbf{x}$. Further, Γ is the diagonal matrix containing the BatchNorm scaling parameters γ and $\boldsymbol{\beta}$ is the vector containing the BatchNorm shifting parameters β .

In the second one, the normalization with batch mean and variance is done before applying the affine transformation with the layer's weight parameters:

$$f_{\Gamma,\beta}(\mathbf{x}) = \sigma(\Gamma(\epsilon\mathbf{I} + \Sigma_{\text{post}})^{-1}(\mathbf{W}^\top \mathbf{x} + \mathbf{b} - \boldsymbol{\mu}_{\text{post}}) + \boldsymbol{\beta}) \quad (9.3)$$

Consequently, the mean $\boldsymbol{\mu}_{\text{post}}$ and variance Σ_{post} here are those of the $(\mathbf{W}^\top \mathbf{x} + \mathbf{b})$. He et al. (2016b) showed that the latter leads to better accuracy. In this paper, we consider scaling transformations of the *second* type.

9.3 Main Results

In this work we study the expressive power of the normalization layers of a frozen or randomly initialized neural network. All of our theorems concern the *expressive power* of such a neural network and don't involve any training. As a result, the mean and variance (calculated over a batch size) are considered to be constants, as they are at inference time. However, we mention here that if one wanted to consider variable weight matrices, means and variances, then we could imagine a sequence of frames/instances with each instance representing a stage of the training process. Then for each of these instances we could apply our results.

We assume that there exists some target network $g(\mathbf{x})$ that has normalization layers; then we prove that there exists a choice of the parameters of the normalization layers of another randomly initialized neural network $f(\mathbf{x})$, such that $f(\mathbf{x}) \equiv g(\mathbf{x})$.

Our first theorem states that a randomly initialized ReLU network with a factor of d overparameterization can reconstruct the target ReLU network g .

Theorem 9.5. *Let $g(\mathbf{x}) = \Gamma_l^* \mathbf{W}_l^* \sigma(\Gamma_{l-1}^* \mathbf{W}_{l-1}^* \sigma(\dots \sigma(\Gamma_1^* \mathbf{W}_1^* \mathbf{x} + \beta_1^*) + \beta_{l-1}^*) + \beta_l^*$ be any ReLU network with depth l , where $\mathbf{x} \in \mathbb{R}^d$, $\Gamma_i^* \in \mathbb{R}^{d \times d}$ and $\mathbf{W}_i^* \in \mathbb{R}^{d \times d}$ with $\|\mathbf{W}_i^*\| \leq 1$ for all $i = 1, \dots, l$. Consider a randomly initialized ReLU network of depth $2l$ with normalization layers:*

$$f(\mathbf{x}) = \Gamma_{2l} \mathbf{W}_{2l} \sigma(\Gamma_{2l-1} \mathbf{W}_{2l-1} \sigma(\dots \sigma(\Gamma_1 \mathbf{W}_1 \mathbf{x} + \beta_1)) + \beta_{2l-1}) + \beta_{2l}$$

where $\Gamma_i \in \mathbb{R}^{d^2 \times d^2}$, $\mathbf{W}_i \in \mathbb{R}^{d \times d^2}$ for all odd $i \in [2l]$ and $\Gamma_i \in \mathbb{R}^{d \times d}$, $\mathbf{W}_i \in \mathbb{R}^{d^2 \times d}$ for all even $i \in [2l]$. Then one can compute the normalization layer parameters $\Gamma_1 \dots \Gamma_{2l}$ and $\beta_1 \dots \beta_{2l}$, so that with probability 1 the two networks are equivalent, i.e., $f(\mathbf{x}) = g(\mathbf{x})$ for all $\mathbf{x} \in \mathbb{R}^d$ with $\|\mathbf{x}\| \leq 1$.

This result shows that the expressive power of scaling and shifting transformations of random features is indeed non-trivial. Recent work by Wang et al. (2021b), and Kamalakara et al. (2022) has shown experimentally that the weight matrices of a neural network can be factorized to low rank ones and then trained with little to no harm in the accuracy of

the model. While in Theorem 9.5, f needs a width overparameterization of the order of d as compared to g , we show that if the weight matrices of the target neural network are in fact factorized and have ranks r each, then f only needs a width overparameterization of the order r as compared to g . Consider the network g' which is created by factorizing the weight matrices of g . Since this can be viewed as a network width depth $2l$, we can apply Theorem 9.5 on g' to get the following corollary.

Corollary 9.6. *Consider a randomly initialized four layer ReLU network with normalization layers and width dr ; then by appropriately choosing the scaling and shifting parameters of the normalization layers, the network can be functionally equivalent with any one layer network of the same architecture, width d and weight matrix of $r < d$ with probability 1.*

The results we presented so far require the width of f to be larger than the width of g . This logically leads to the following question: *Can making f deeper help reduce its width?* Frankle et al. (2020c) showed experimentally that increasing depth, while keeping width fixed, increases the expressive power of normalization layer parameters. Turns out that this is indeed possible by leveraging skip connections. The following theorem states that any fully connected neural network can be realized by a deeper, randomly initialized neural network with skip connections by only tuning its normalization layer parameters.

Theorem 9.7. *Consider the following neural network*

$$g(\mathbf{x}) = \mathbf{\Gamma}_l^* \mathbf{W}_l^* \sigma(\dots \sigma(\mathbf{\Gamma}_1^* \mathbf{W}_1^* \mathbf{x} + \beta_1^*)) + \beta_l^*$$

where $\mathbf{\Gamma}_i^* \in \mathbb{R}^{d \times d}$, $\mathbf{W}_i^* \in \mathbb{R}^{d \times d}$ with $\|\mathbf{W}_i^*\| \leq 1$ for all $i = 1, \dots, l$. Let the $j(\lceil \frac{d}{k} \rceil + 1)$ -th layer of the neural network f be

$$f^j(\mathbf{z}) = \sigma(\mathbf{W}_{\lceil \frac{d}{k} \rceil + 1}^j L_{\lceil \frac{d}{k} \rceil}^j) \text{ for } j = 1, \dots, l,$$

where, $\mathbf{z} \in \mathbb{R}^d$,

$$\begin{aligned}
\mathbf{W}_{\lceil \frac{d}{k} \rceil + 1}^j &\in \mathbb{R}^{d \times dk}, \text{ for } j = 1, \dots, l \\
L_1^j &= \mathbf{\Gamma}_1^j \mathbf{W}_1^j \mathbf{z}_1^j + \beta_1^j, \\
L_i^j &= \mathbf{\Gamma}_i^j \mathbf{W}_i^j (\sigma(L_{i-1}^j) + \mathbf{A}_i^j \mathbf{z}_i^j) + \beta_i^j \text{ for } i = 2, \dots, \lceil \frac{d}{k} \rceil \\
\mathbf{W}_i^j, \mathbf{\Gamma}_i^j &\in \mathbb{R}^{dk \times dk}, \mathbf{A}_i^j \in \mathbb{R}^{dk \times k}, \text{ and } \beta_i^j \in \mathbb{R}^{dk} \text{ for all } i = 1, \dots, \lceil \frac{d}{k} \rceil, j = 1, \dots, l.
\end{aligned}$$

Here the matrices $\mathbf{W}_i^j, \forall i, j$ are randomly initialized and then frozen. Then, with probability 1, one can compute normalization layer parameters so that the two networks are functionally equivalent:

$$g(\mathbf{x}) = f^l(f^{l-1}(\dots f^1(\mathbf{x}))), \forall \|\mathbf{x}\| \leq 1 \quad (9.4)$$

Here, the parameter k is tunable and can be any integer in $[d]$.

Note that the parameter k can be used to achieve a trade-off between width and depth. If k is increased, then the width increases, but the depth decreases and vice-versa. Further for the network f , note that each matrix in Theorem 9.5 has dimensions $d^2 \times d$ or $d \times d^2$, for a total of $\mathcal{O}(d^3 l)$ parameters in the network, while in Theorem 9.7 we have matrices of dimensions $d \times dk$ or $dk \times dk$. If we set $k = 1$ the width of f matches that of g , however it can be checked that the total number of parameters in f still remains $\mathcal{O}(d^3 l)$, which is the same as that from Theorem 9.5.

Remark 9.8. Theorem 9.5 and Theorem 9.7 assume that all layers of g have width d , the same as input dimension. However, all the results in the paper can be extended to the general case where all the layers of g can have different widths. We omit that setting for ease of exposition.

Our last result concerns the total number of parameters that the model entails. The question we tackle here is the sparsification of the random matrices of the network in Theorem 9.5. Formally,

Theorem 9.9. For the setting of Theorem 9.5, consider that the random matrices of each of the layers are sparsified with probability $p = \Theta(\sqrt{\log d/d})$, meaning that each of their elements is zero

with probability $1 - p$. If the depth of the network is polynomial in the input, i.e., $l = \text{poly}(d)$ then the results of Theorem 9.5 hold with probability at least $1 - 1/d$.

Remark 9.10. This sparsification results in a total number of $\tilde{O}(d^2\sqrt{d})$ non-zero parameters in f . However, we don't think that this result is tight. We believe that it can be improved to a higher sparsity of $p = \Theta(\log d/d)$, which would result in the total number of non-zero parameters being $\tilde{O}(d^2)$.

9.4 Our Techniques

In general, the derivation of our results rely heavily on the invertibility of the Khatri-Rao product, the establishment of full-rankness of matrix multiplications and non-degeneracy of them, by exploiting the randomness of the weight matrices.

9.4.1 Reconstruction with overparameterization

We prove our first result, Theorem 9.5, by building a layer-by-layer reconstruction of g : For the i -th layer of g , we construct the $2i - 1$ and $2i$ layers of f . We use the shifting parameters β_{2i-1} of f to activate all the ReLUs of the $2i - 1$ -th layer of f and then use the shifting parameters of the next layer, β_{2i} , to cancel out any extra bias introduced by β_{2i-1} . Finally, we use Γ_{2i-1} to reconstruct the targeted matrix. Γ_{2i} could be set to any arbitrary value (full-rank) diagonal matrix, but for the sake of convenience, we just set it to be the identity matrix.

Proof Sketch. We prove that the first layer of g and the first two layers of f are functionally equivalent. Without loss of generality, the same proof can be applied to all subsequent layers of g and f . Thus, it is sufficient to show that $\Gamma_2 \mathbf{W}_2 \sigma(\Gamma_1 \mathbf{W}_1 \mathbf{x} + \beta_1) + \beta_2 = \mathbf{W}_1^* \mathbf{x}$ for all $\mathbf{x}$ with $\|\mathbf{x}\| \leq 1$.

Step 1. We set the parameters of β_1 large enough so that all ReLUs of the first layer of f are activated. For this, we only need that the parameters of the first normalization layer satisfy $\beta_1^i = \Omega(|\gamma_1^i|)$ ¹ for each neuron i . Next, we simply set $\Gamma_2 := \mathbf{I}_d$ and $\beta_2 := -\mathbf{W}_2\beta_1$. Substituting these we get that $\Gamma_2\mathbf{W}_2\sigma(\Gamma_1\mathbf{W}_1\mathbf{x} + \beta_1) + \beta_2 = \mathbf{W}_2\Gamma_1\mathbf{W}_1\mathbf{x}$. Thus, now we need to show that the system $\mathbf{W}_2\Gamma_1\mathbf{W}_1\mathbf{x} = \mathbf{W}_1^*\mathbf{x}$ has a solution with respect to Γ_1 . As we will see, this solution will, in fact, be unique.

Step 2. We use the following lemma to construct a Γ_1 that satisfies the equality above.

Lemma 9.11. *Let $\mathbf{C} \in \mathbb{R}^{n \times m}$, $\mathbf{X} \in \mathbb{R}^{nm \times nm}$ be a diagonal matrix, $\mathbf{B} \in \mathbb{R}^{nm \times m}$ and $\mathbf{W}^* \in \mathbb{R}^{n \times m}$. Then, the following holds for the equalities below*

$$\mathbf{CXB} = \mathbf{W}^* \iff (\mathbf{C} \odot \mathbf{B}^T)_{\text{vecd}}(\mathbf{X}) = \text{vec}(\mathbf{W}^*),$$

where vecd denotes the vectorization of the diagonal of a matrix, and vec denotes the row major vectorization of a matrix.

The lemma above implies that if $\mathbf{W}_2 \odot \mathbf{W}_1^T$ is invertible, then we can find unique Γ_1 ² that satisfies $\mathbf{W}_2\Gamma_1\mathbf{W}_1 = \mathbf{W}_1^*$. The following lemma says that with probability 1, this is true.

Lemma 9.12. *Let $\mathbf{A} \in \mathbb{R}^{n \times nm}$, $\mathbf{B} \in \mathbb{R}^{m \times nm}$ be two random matrices, whose elements are drawn independently from a continuous distribution; then, their Khatri-Rao product, $\mathbf{A} \odot \mathbf{B}$, is full rank with probability 1.*

For some intuition regarding the proof of this lemma, consider $\mathbf{A}$ to be an $nm \times nm$ matrix, whose elements are drawn from any continuous distribution independently. It is easy to see that this matrix is full rank, since the columns of this matrix are independent vectors, and the entries of vectors themselves come from a product distribution over $\mathbb{R}^{nm}$.

¹See the proof of Theorem E.18 for the exact constant.

²Since the matrix $\mathbf{W}_2 \odot \mathbf{W}_1$ is full rank, the vector $\text{vec}(\mathbf{W}^*)$ will be a linear combination of its columns; from the well-known theorem of Rouché-Kronecker-Capelli (see Theorem E.8) the solution will be unique.

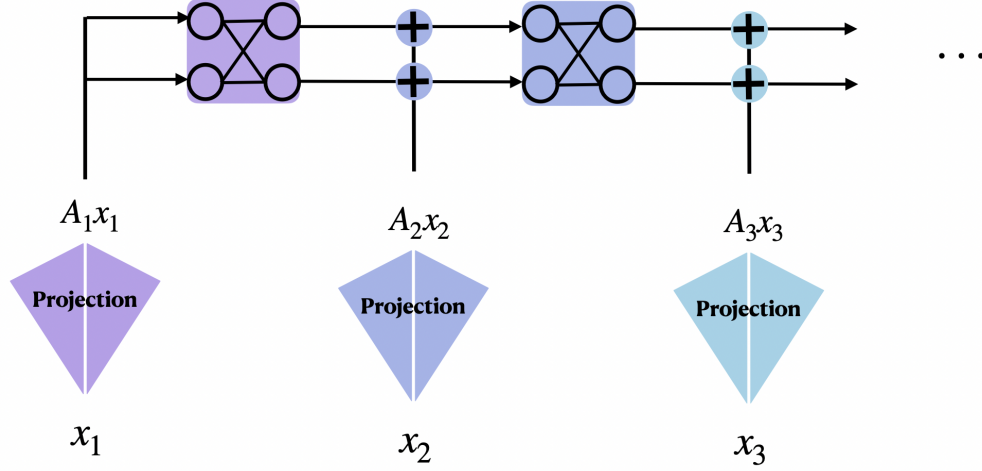


Figure 9.2: A visualization of the skip connections. x_i is the i -th partition of the input x , which is then projected through the matrices A_i to correct its dimension.

For the case of the Khatri-Rao product $A \odot B$, where $A \in \mathbb{R}^{n \times nm}$ and $B \in \mathbb{R}^{m \times nm}$ however, we only have $(n + m)nm$ ‘free’ random variables, instead of $(nm)^2$. However, writing down the expression for the determinant as a polynomial in the random variables, we see that certain terms have independent coefficients from others. This helps us prove that the probability of the determinant being zero is zero.

Step 3. Repeating the same proof above for all the layers of g and the corresponding layers of f proves a layer-wise equivalence between f and g . Since f and g are just the composition of these layers, proving the equivalence layer-wise proves that $f \equiv g$.

9.4.2 Width/Depth tradeoff

The architecture we consider is as follows (see Fig. 9.2): Let $x \in \mathbb{R}^d$ denote the input to the layer i of g . Then for constructing the corresponding layers of f which are functionally equivalent to the the layer i in g , we first partition $x \in \mathbb{R}^d$ into blocks of size k , which we denote with $x_1, \dots, x_{\lceil \frac{d}{k} \rceil}$. Each of these blocks is then passed to a different layer through the skip connections, with each layer having width dk .

At the end of $\lceil \frac{d}{k} \rceil$ layers, a final linear layer is added to “correct” the dimensions and

match them with the ones of the target network. This new architecture is applied to substitute each of the layers of the target network, as is illustrated in Fig. 9.1.

Skip connections. As explained earlier, we break the input $\mathbf{x}$ into chunks of size k . Thus for example the first chunk, $\mathbf{x}_1 = [x_1, \dots, x_k]$. This can be seen as a projection to a k dimensional subspace. However, as we mentioned each layer of this construction will have dimension dk , we thus use now a random matrix $\mathbf{A}_i \in \mathbb{R}^{k \times dk}, i = 1, \dots, \lceil \frac{d}{k} \rceil$ to project the input and match the dimensions.

Proof Sketch. The idea is to partition the input, pass a different piece of the input in each individual layer and try to exactly reconstruct the parts of the target matrix $\mathbf{W}^*$ that correspond to the specific piece. Let k be the size of each partition and so in total we need to have $\lceil \frac{d}{k} \rceil$ layers. More specifically, we can rewrite $\mathbf{W}^*$ as

$$\mathbf{W}^* = \mathbf{W}^* \mathbf{S}_1^T \mathbf{S}_1 \mathbf{x} + \mathbf{W}^* \mathbf{S}_2^T \mathbf{S}_2 \mathbf{x} + \dots + \mathbf{W}^* \mathbf{S}_{\lceil \frac{d}{k} \rceil}^T \mathbf{S}_{\lceil \frac{d}{k} \rceil} \mathbf{x} \quad (9.5)$$

where $\mathbf{S}_i \in \mathbb{R}^{k \times d}$ is a sub-selection matrix and specifically $\mathbf{S}_i = \begin{bmatrix} \mathbf{0} & \mathbf{I}_k & \mathbf{0} \end{bmatrix}$ where the $k \times k$ identity matrix occupies the $(i-1)k+1, \dots, ik$ columns.

Having this in mind, we will use each one of the layers with the skip connections to approximate a specific part of the input, which is then projected to a dk -dimensional vector through a random matrix $\mathbf{A}_i$ to match the dimensions of the input. The output of the Neural Network is

$$f(\mathbf{x}) = \mathbf{W}_{\lceil \frac{d}{k} \rceil + 1} L_{\lceil \frac{d}{k} \rceil} \quad (9.6)$$

where $L_1 = \mathbf{\Gamma}_1 \mathbf{W}_1 \mathbf{x}_1 + \beta_1$ and $L_i = \mathbf{\Gamma}_i \mathbf{W}_i [\sigma(L_{i-1}) + \mathbf{A}_i \mathbf{x}_i] + \beta_i$ for $i = 2, \dots, \lceil \frac{d}{k} \rceil - 1$.

Linearization. We will set the parameters $\beta_i, i = 1, \dots, \lceil \frac{d}{k} \rceil - 1$ to be such that all ReLUs are activated as we did in the proof of Theorem 9.5, while we set $\beta_{\lceil \frac{d}{k} \rceil}$ appropriately to cancel out any error created.

Segmentation. After the previous step has been completed, the output of the network becomes

$$f(\mathbf{x}) = \mathbf{W}_{\lceil \frac{d}{k} \rceil + 1} \left(\prod_{i=1}^{\lceil \frac{d}{k} \rceil} \Gamma_i \mathbf{W}_i \mathbf{A}_1 \mathbf{x}_1 + \prod_{i=2}^{\lceil \frac{d}{k} \rceil} \Gamma_i \mathbf{W}_i \mathbf{A}_2 \mathbf{x}_2 + \dots + \Gamma_{\lceil \frac{d}{k} \rceil} \mathbf{W}_{\lceil \frac{d}{k} \rceil} \mathbf{A}_{\lceil \frac{d}{k} \rceil} \mathbf{x}_{\lceil \frac{d}{k} \rceil} \right) \quad (9.7)$$

We now use each of the terms in the summation above to reconstruct the corresponding term in Eq. (9.5). Thus we need to solve a system of $\lceil \frac{d}{k} \rceil$ linear systems. We show that this system is indeed feasible, by using induction.

Induction. For the base case, we start with the last equation and we show that the linear system

$$\mathbf{W}_{\lceil \frac{d}{k} \rceil + 1} \Gamma_{\lceil \frac{d}{k} \rceil} \mathbf{W}_{\lceil \frac{d}{k} \rceil} \mathbf{A}_{\lceil \frac{d}{k} \rceil} = \mathbf{W}_{\lceil \frac{d}{k} \rceil}^* \quad (9.8)$$

has a unique solution with probability 1. This is a corollary of the proof of Theorem 9.5.

Once the solution for this equation has been found, the idea is to move on to the next layer. One key detail is that we need to prove that all elements of $\Gamma_{\lceil \frac{d}{k} \rceil}$ will be non-zero with probability one. This is because $\Gamma_{\lceil \frac{d}{k} \rceil}$ and any Γ_i appear in Eq. (9.7) the products corresponding to all previous layers. Thus, if one of the elements of these matrices is zero, full-rankness and hence a solution to the corresponding linear systems cannot be guaranteed.

Lemma 9.13. *Let $\mathbf{A} \in \mathbb{R}^{n \times nm}$, $\mathbf{C} \in \mathbb{R}^{m \times nm}$ be random matrices, $\mathbf{D} \in \mathbb{R}^{nm \times nm}$ a diagonal fixed matrix with non-zero elements and w^* a fixed vector which is not the zero vector. Define $\mathbf{B} = \mathbf{A} \odot (\mathbf{C}\mathbf{D})$, then*

$$\mathbb{P}(\langle \mathbf{B}_i^{-1}, w^* \rangle = 0) = 0 \quad (9.9)$$

where $\mathbf{B}_i^{-1}$ is the i -th row of the inverse of $\mathbf{B}$.

For the inductive step, we assume that all components up to (and not included) the $i - th$ component have been exactly reconstructed and $\Gamma_j, j = \lceil \frac{d}{k} \rceil, \dots, i - 1$ have non-zero

elements with probability one; and then we reconstruct the i -th component and show that Γ_i also has only non-zero diagonal elements with probability one.

9.4.3 Reconstruction of Sparse Networks

The proof of this result has two components to it. Our target is to show that if the matrices involved in the Khatri-Rao product appeared in Section 9.4.1 are sparsified randomly, then with high probability the matrix created is still full rank. We first show that as long as both matrices have i.i.d. entries from a continuous distribution, the sparsity pattern that is created dictates whether the matrix is non-singular or not. Then, we prove bounds on the non-singularity of Khatri-Rao products of random Boolean matrices by reducing the problem to the problem of non-singularity of random Boolean matrices with i.i.d. elements.

Proof Sketch. We now provide more details of the steps we follow to prove this result. The sparsification of each of the random weight matrices can be expressed as their Hadamard product with a random Bernoulli matrix, *i.e.*, $(\mathbf{W} \circ \mathbf{M})$, where $\mathbf{W}$ has each entry i.i.d. from a continuous distribution and $\mathbf{M}$ is a Boolean matrix. From the proof technique presented in Section 9.4.1, we conclude that it is sufficient to show that the probability that the matrix $(\mathbf{W}_1 \circ \mathbf{M}_1) \odot (\mathbf{W}_2 \circ \mathbf{M}_2)$ is *not* invertible is very small. We assume that the elements of $\mathbf{M}_1, \mathbf{M}_2$ are *i.i.d.* $\text{Bern}(p)$ ³. Our target is to find a value of the probability p that ensures small enough probability of non-invertibility of the Khatri-Rao product.

Step 1. As mentioned above, we will relate the invertibility of $(\mathbf{W}_1 \circ \mathbf{M}_1) \odot (\mathbf{W}_2 \circ \mathbf{M}_2)$ to the invertibility of $\mathbf{M}_1 \odot \mathbf{M}_2$. To do so, we first introduce the notion of the *Boolean determinant* for a Boolean matrix.

Definition 9.14. *The Boolean determinant of a Boolean matrix $\mathbf{B} \in \mathbb{R}^{d \times d}$ is recursively defined as follows:*

³We say that $x \sim \text{Bern}(p)$, if $\mathbb{P}(x = 1) = p$ and $\mathbb{P}(x = 0) = 1 - p$

- if $d = 1$, $\text{Det}_{\text{Bool}}(\mathbf{B}) := B_{1,1}$, that is, it is the only element of the matrix, and
- if $d > 1$, $\text{Det}_{\text{Bool}}(\mathbf{B}) := \max_{i \in \{1, \dots, d\}} (B_{i,1} \text{Det}_{\text{Bool}}(\mathbf{B}_{-(i,1)}))$.

where $B_{i,j}$ is the element of $\mathbf{B}$ at i -th row and j -th column, and $\mathbf{B}_{-(i,j)}$ is the sub-matrix of $\mathbf{B}$ with i -th row and j -th column removed. Note that $\text{Det}_{\text{Bool}}(\mathbf{B}) \in \{0, 1\}$ for any Boolean matrix $\mathbf{B}$.

We start by proving the following:

$$\text{Det}((\mathbf{W}_1 \circ \mathbf{M}_1) \odot (\mathbf{W}_2 \circ \mathbf{M}_2)) = 0 \Leftrightarrow \text{Det}_{\text{Bool}}(\mathbf{M}_1 \odot \mathbf{M}_2) = 0$$

To prove this, note that despite the fact that the Khatri-Rao product matrix $(\mathbf{W}_1 \circ \mathbf{M}_1) \odot (\mathbf{W}_2 \circ \mathbf{M}_2)$ no longer has independent entries, its columns are still independent. Hence, while writing the expression for determinant computed along any column, the corresponding sub-determinants are independent of the column. This combined with the fact that the entries of $\mathbf{W}_1$ and $\mathbf{W}_2$ come from continuous distributions, can be used to show the equivalence above.

Step 2. Thus, next we need to prove that the Boolean determinant of $\mathbf{M}_1 \odot \mathbf{M}_2$ is non-zero with a small probability. In order to do so, we start replacing each column of $\mathbf{M}_1 \odot \mathbf{M}_2$ with a new vector of equal length, which has $\text{Bern}(q)$ i.i.d. elements. Let this new matrix be $\mathbf{M}$. We want to find the value of q as a function of p such that the following holds:

$$\mathbb{P}(\text{Det}_{\text{Bool}}(\mathbf{M}_1 \odot \mathbf{M}_2) = 0) \leq \mathbb{P}(\text{Det}_{\text{Bool}} \mathbf{M} = 0).$$

Once we find such a q , we can keep replacing each column of $\mathbf{M}_1 \odot \mathbf{M}_2$ to get a matrix $\mathbf{M}'$ which all of its entries are i.i.d. $\text{Bern}(q)$ variables, and for which $\mathbb{P}(\text{Det}_{\text{Bool}}(\mathbf{M}_1 \odot \mathbf{M}_2) = 0) \leq \mathbb{P}(\text{Det}_{\text{Bool}} \mathbf{M}' = 0)$. We can then use existing results on the singularity of Boolean matrices with i.i.d. entries.

Step 3. We show that if $p \geq \sqrt{2qd}$ then the requirement above is satisfied. To prove this, we first show that it is sufficient to prove the following, stronger statement: If $\mathbf{u}_0, \dots, \mathbf{u}_{d-1}, \mathbf{v}_0, \dots, \mathbf{v}_{d-1} \sim \text{Bern}(p)$ and are sampled independently and $\mathbf{x}_0, \dots, \mathbf{x}_{d^2-1} \sim \text{Bern}(q)$ are also independent with $p \geq \sqrt{2qd}$; then, for any subset $\mathcal{S} \subset \{0, \dots, d^2 - 1\}$,

$$\mathbb{P} \left(\sum_{i \in \mathcal{S}} \mathbf{u}_{\lfloor i/d \rfloor} \mathbf{v}_{i \pmod{d}} = 0 \right) \leq \mathbb{P} \left(\sum_{i \in \mathcal{S}} \mathbf{x}_i = 0 \right)$$

This proof requires probabilistic arguments and is deferred to the Appendix.

Step 4. After this result has been established, we use the following theorem to get bounds on probability of non-singularity of the i.i.d. Boolean entries matrix created in the previous step:

Theorem 9.15 (Theorem 1.1 in (Basak and Rudelson, 2018)). *Let $\mathbf{A} \in \mathbb{R}^{d \times d}$ be an a matrix with i.i.d. $\text{Ber}(p)$ entries. Then, there exist absolute constants $0 < c, \bar{c}, C < \infty$ such that for any $\epsilon > 0$, and ρ such that $n\rho \geq \log(1/\rho)$, we have*

$$\mathbb{P} \left(\left\{ \left(s_{\min}(\mathbf{A}) \leq \bar{c}\epsilon \exp \left(-C \frac{\log(1/\rho)}{\log(n\rho)} \sqrt{\frac{\rho}{n}} \right) \right) \cap \Omega_o^c \right\} \right) \leq \epsilon + n^{-c}$$

where $s_{\min}$ is the minimum singular value of the matrix $\mathbf{A}$ and Ω_o is the event that any column or some row is identical to 0.

It is easy to see that $n\rho \geq \log(1/\rho)$ holds when $\rho \geq \log n/n$. For the purpose of our work, we will focus on the invertibility of matrix $\mathbf{A}$, that is the probability that $s_{\min}(\mathbf{A}) > 0$. As a result, let ϵ be arbitrarily small and $\rho = \log n/n$; then the matrix $\mathbf{A}$ is non-singular with probability of the order $1/n^c$. We emphasize that the constant c is universal.

Using Theorem 9.15, we get that every Bernoulli i.i.d. matrix $\text{Bern}(\rho)$, with $\rho = \log d^2/d^2$, is singular with probability at most d^{-2c} for a universal constant c . However for our purposes, we need the probability of being singular to be $d^{-\bar{C}}$ for some $\bar{C}$ which we will determine later in the proof. To achieve this smaller probability, consider drawing

independently $\frac{\bar{C}}{c}$ Bern(ρ) matrices: $\mathbf{E}_i, \forall i \in [\frac{\bar{C}}{c}]$. Then the probability of all of them being singular is at most $d^{-2\bar{C}}$. It is straightforward to see that whenever the determinant of a Boolean matrix is non-zero then the Boolean Determinant of that matrix is also non-zero. Hence, with probability at least $1 - d^{-2\bar{C}}$, at least one of these matrices has non-zero Boolean determinant. We create our matrix $\mathbf{M}' = \mathbf{E}_1 \vee \dots \vee \mathbf{E}_{\frac{\bar{C}}{c}}$, where the 'or' operation is done element-wise. Note that $\mathbf{M}'$ has a non-zero Boolean determinant if any of the matrices $\mathbf{E}_1 \dots \mathbf{E}_{\frac{\bar{C}}{c}}$ has a non-zero one. Thus $\mathbf{M}'$ has a non-zero Boolean determinant with probability at least $1 - d^{-2\bar{C}}$. Note that $\mathbf{M}'$ is itself an i.i.d. Bernoulli matrix, with $q \approx \rho\bar{C}/c = \mathcal{O}(\log d^2/d^2)$.

Step 5. To conclude, since $\mathbf{M}'$ has a zero Boolean determinant with probability at most $d^{-2\bar{C}}$, then to ensure that the Khatri-Rao product of the sparse matrices is invertible for all layers, we need to take a union bound over all the layers. Thus, the probability that our result will not hold is at most $d^{-2\bar{C}}2l$. Since we have assumed that $l = \mathcal{O}(\text{poly}(d))$, for large enough constant $\bar{C}$, the overall probability of the union bound can be driven down to be at most $1/d$. Recalling that $p = \sqrt{2dq}$, we get that $p = \Theta(\sqrt{\log d/d})$ suffices for the theorem to hold.

9.5 Discussion and Future Work

This work focuses on the expressive power of normalization parameters; but although our methods are constructive, it is important to note that the results may not apply to the training process through gradient based optimization algorithms. It would be interesting though to see if techniques like SGD can indeed leverage this power to the fullest. We also note that our approach is limited, in the sense that we aimed for exact reconstruction of the target network, and which necessarily meant that we need d^2 normalization parameters per layer. The regime in which normalization layer is only moderately overparameterized could be explored in future work. Another interesting direction is the fine-tuning capacity

of normalization layers, where we are given a network trained on a certain task, and we want to fine-tune it on a slightly different task by only modifying the normalization layers. Extending our results to other architectures like CNNs and Transformers, and proving lower bounds or impossibility results about the expressive power of normalization layers are also exciting open problems.

10 LOGARITHMIC BOUNDS FOR STRONG LTH VIA THE SUBSET SUM

PROBLEM

The strong *lottery ticket hypothesis* (LTH) postulates that one can approximate any target neural network by only pruning the weights of a sufficiently over-parameterized random network. A recent work by Malach et al. (2020b) establishes the first theoretical analysis for the strong LTH: one can provably approximate a neural network of width d and depth l , by pruning a random one that is a factor $O(d^4 l^2)$ wider and twice as deep. This polynomial over-parameterization requirement is at odds with recent experimental research that achieves good approximation with networks that are a small factor wider than the target. In this work, we close the gap and offer an exponential improvement to the over-parameterization requirement for the existence of lottery tickets. We show that any target network of width d and depth l can be approximated by pruning a random network that is a factor $O(\log(dl))$ wider and twice as deep. Our analysis heavily relies on connecting pruning random ReLU networks to random instances of the SUBSETSUM problem. We then show that this logarithmic over-parameterization is essentially optimal for constant depth networks. Finally, we verify several of our theoretical insights with experiments.

10.1 Introduction

Many of the recent unprecedented successes of machine learning can be partially attributed to state-of-the-art neural network architectures that come with up to tens of billions of trainable parameters. Although test accuracy is one of the gold standards in choosing one of these architectures, in many applications having a “compressed” model is of practical interest, due to typically reduced energy, memory, and computational footprint Deng et al. (2020); Han et al. (2015b, 2016); Li et al. (2016); Wen et al. (2016); Hubara et al. (2016, 2017);

He et al. (2017); Wu et al. (2016); Zhu et al. (2016); He et al. (2018); Zhu and Gupta (2017); Cheng et al. (2019); Blalock et al. (2020). Such a compressed form can be achieved by either modifying the architecture to be leaner in terms of the number of weights, or by starting with a high-accuracy network and pruning it down to one that is sparse in some representation domain, while not sacrificing much of the original network’s accuracy. A rich and long body of research work shows that one can prune a large network to a tiny fraction of its size, while maintaining (or sometimes even improving) its original accuracy Deng et al. (2020); Han et al. (2015b, 2016); Li et al. (2016); Wen et al. (2016); Hubara et al. (2016, 2017); He et al. (2017); Wu et al. (2016); Zhu et al. (2016); He et al. (2018); Zhu and Gupta (2017); Cheng et al. (2019); Blalock et al. (2020).

Although network pruning dates back at least to the 80s Hassibi and Stork (1993a); Levin et al. (1994); Mozer and Smolensky (1989a); LeCun et al. (1990), there has been a recent flurry of results that introduce sophisticated pruning, sparsification, and quantization techniques which lead to significantly compressed model representations that attain state-of-the-art accuracy Deng et al. (2020); Cheng et al. (2019); Blalock et al. (2020). Several of these pruning methods require many rounds of pruning and retraining, resulting in a time-consuming and hard to tune iterative meta-algorithm.

Could we avoid this pruning and retraining cycle, by “winning” the weight initialization lottery, with a ticket that puts stochastic gradient descent (SGD) on a path to, not only accurate, but also very sparse neural networks? Frankle and Carbin Frankle and Carbin (2018b) define *lottery tickets* as sparse subnetworks of a randomly initialized network, that if trained to full accuracy just once, can reach the performance of the fully-trained but dense target model. If these lottery tickets can be found efficiently, then the computational burden of pruning and retraining can be avoided.

Several works build and expand on the *lottery ticket hypothesis* (LTH) that was introduced by Frankle and Carbin Frankle and Carbin (2018b). Zhou et al. Zhou et al. (2020) experimentally analyze different strategies for pruning. Frankle et al. Frankle et al. (2020b)

relate the existence of lottery tickets to a notion of stability of networks for SGD. Cosentino et al. Cosentino et al. (2019) show that lottery tickets are amenable to adversarial training, which can lead to both sparse and robust neural networks. Soelen et al. Soelen and Sheppard (2019) show that the winning tickets from one task are transferable to another related task. Sabatelli et al. Sabatelli et al. (2020) show further that the trained winning tickets can be transferred with minimal retraining on new tasks, and sometimes may even generalize better than models trained from scratch specifically for the new task.

Along this literature, a striking finding was reported by Ramanujan et al. Ramanujan et al. (2020) and Wang et al. Wang et al. (2019): one does not even need to train the lottery tickets to get high test accuracy; they find that high-accuracy, sparse models simply reside within larger random networks, and appropriate pruning can reveal them. However, these high-accuracy lottery tickets do not exist in plain sight, and finding them is in itself a challenging (as a matter of fact NP-Hard) computational task. Still, the mere existence of these random substructures is interesting, and one may wonder whether it is a universal phenomenon.

The phenomenon corroborated by the findings of Ramanujan et al. Ramanujan et al. (2020) is referred to as the *strong lottery ticket hypothesis*. Recently, Malach et al. Malach et al. (2020b) proved the strong LTH for fully connected networks with ReLU activations. In particular, they show that one can approximate any target neural network, by pruning a sufficiently over-parameterized network of random weights. The degree of over-parameterization, i.e., how much larger this random network has to be, was bounded by a polynomial term with regards to the width d and depth l of the target network. Specifically, their analysis requires the random network to be of width $\tilde{O}(d^5 l^2 / \epsilon^2)$ and depth $2l$, to allow for a pruning that leads to an ϵ -approximation with regards to the output of the target network, for any input in a bounded domain. They also show that the required width can be improved to $\tilde{O}(d^2 l^2 / \epsilon^2)$ under some sparsity assumptions on the input. Although polynomial, the required degree of over-parameterization is still too demanding, and it

is unclear that it explains the experimental results that corroborate the strong LTH. For example, it does not reflect the findings in Ramanujan et al. (2020) that only seem to require a constant factor over-parameterization, e.g., a randomly initialized Wide ResNet50, can be pruned to a model that has the accuracy of a fully trained ResNet34. In this work, our goal is to address the following question:

What is the required over-parameterization so that a network of random weights can be pruned to approximate a smaller target network?

Towards this goal, we identify the crucial step in the proof strategy of Malach et al. (2020b) that leads to the polynomial factor requirement on the per-layer over-parameterization. Say that one wants to approximate a weight $w^* \in [-1, 1]$ with a random number drawn from a distribution, e.g., $\text{Uniform}([-1, 1])$. If we draw $n = O(1/\epsilon)$ i.i.d. samples,

$$X_1, \dots, X_n \sim \text{Uniform}([-1, 1]),$$

then one of these X_i 's will be ϵ close to w^* , with constant probability. In a way, this random sampling generates an ϵ -net, i.e., a set of numbers such that any weight in $[-1, 1]$ is ϵ close to one of these n samples. Pruning that set down to a single number, i.e., selecting the point closest to w^* , leads to an ϵ approximation for a single weight. At the cost of a polynomial overhead on the number of samples n , one can appropriately apply this to every weight of a given layer, and then all layers of the network, in order to obtain a uniform approximation result for every possible input of the neural network.

Perhaps the surprising fact is that one can achieve the same approximation, with exponentially smaller number of samples, while still using a pruning algorithm. The idea is that instead of approximating the target weight w^* with only one of the n samples $X_1, \dots, X_n$, one could add a subset of them to get a better approximation. Indeed, when

$n = O(\log(1/\epsilon))$, then, with high probability, there exists a subset $S \subseteq [1, \dots, n]$, such that

$$\left| w^* - \sum_{i \in S} X_i \right| \leq \epsilon.$$

Hence, approximating w^* by the best subset sum $\sum_{i \in S} X_i$, offers an exponential improvement on the sample complexity requirement compared to approximating it with one of the X_i 's.

The above approximation result for random X_i 's is drawn from a line of work on the random SUBSETSUM problem Karp (1972); Karmarkar et al. (1986); Lueker (1982, 1998b). In particular, Lueker (1998b) showed that one can achieve an ϵ -approximation of any target number $t \in [-1, 1]$, with only $O(\log(1/\epsilon))$ i.i.d. samples from any distribution that contains a uniform distribution on $[-1, 1]$, by using the solution to the following (NP-hard in general) problem

$$\min_S \left| t - \sum_{i \in S} X_i \right|.$$

Adapting this result to ReLU activation functions is precisely what allows us to get an exponential improvement on the over-parameterization required for the strong LTH to be true.

Our Contributions: In this work, by adapting the random SUBSETSUM results of Lueker (1998b) to ReLU activation functions, we offer an exponential improvement on the over-parameterization required for the strong LTH to be true. In particular we establish the following result.

Theorem 10.1 (Informal). *A randomly initialized network with width $O(d \log \frac{dl}{\min\{\epsilon, \delta\}})$ and depth $2l$, with probability at least $1 - \delta$, can be pruned to approximate any neural network with width d and depth l , up to error ϵ .*

The formal statement of Theorem 10.3 is provided in Section 10.3. Note that in the above result, no training is required to obtain the sparser network within the random model,

i.e., *pruning is all you need*. Further, note that we guarantee a good approximation to *any* network of fixed width and depth by pruning a *single* larger network that is logarithmically wider. That is, the set of networks obtained by pruning the larger random network amounts to an ϵ -net with regards to the smaller target networks.

We then show that this logarithmic over-parameterization is essentially optimal for constant depth networks. Specifically, we provide a lower bound for 2-layered networks that matches the upper bound proposed in Theorem 10.3, up to logarithmic terms with regards to the width (see Theorem 10.8 in Section 10.4 for a formal statement):

Theorem 10.2 (Informal). *There exists a 2-layer neural network with width d which cannot be approximated to error within ϵ by pruning a randomly initialized 2-layer network, unless the random network has width at least $\Omega(d \log(1/\epsilon))$.*

To the best of our knowledge, the only prior work that proves the validity of the strong LTH is Malach et al. Malach et al. (2020b). However, as discussed earlier, their result gives an upper bound of $\tilde{O}(d^5 l^2)$ on the required width of the random network to be pruned, which we improve to $O(d \log(dl))$. A concurrent and independent work by Orseau et al. Orseau et al. (2020) also prove a version of the strong LTH, where they use a hyperbolic distribution for initialization, and require the over-parameterized network to have width $O(d^2 \log(dl))$. Although Orseau et al. Orseau et al. (2020) prove their result for the hyperbolic distribution, they conjecture that it also holds for the uniform distribution. Theorem 10.3 proves this conjecture with an improved guarantee that a $O(d \log(dl))$ wide network suffices (which is smaller by a factor of d). Furthermore, our result holds for a broad class of distributions beyond the uniform distribution (see Remark 10.4).

Notation We use lower-case letters to represent scalars, e.g., we may use w to denote the weight of a single link between two neurons. We use bold lower-case letters to denote vectors, for example, $\mathbf{v}$, $\mathbf{u}$, $\mathbf{v}_1$, $\mathbf{v}_2$. The i -th coordinate of the vector $\mathbf{v}$ is denoted as v_i . Finally, matrices are denoted by bold upper-case letters. For a vector $\mathbf{v}$, we use $\|\mathbf{v}\|$ to denote its

ℓ_2 norm. If a matrix $\mathbf{W}$ has dimension $d_1 \times d_2$, we say $\mathbf{W} \in \mathbb{R}^{d_1 \times d_2}$. The operator norm of a $d_1 \times d_2$ dimensional matrix $\mathbf{M}$ is defined as $\|\mathbf{M}\| = \sup_{\mathbf{x} \in \mathbb{R}^{d_2}, \|\mathbf{x}\|=1} \|\mathbf{M}\mathbf{x}\|$. We denote the uniform distribution on $[a, b]$ by $U[a, b]$. We use c, C to denote positive absolute constants, which may vary from place to place, and their exact values can be inferred from the proof details

10.2 Preliminaries and Setup

In this work, our goal is to approximate a target network $f(\mathbf{x})$ by pruning a larger network $g(\mathbf{x})$, where $\mathbf{x} \in \mathbb{R}^{d_0}$. The target network f is a fully-connected, ReLU neural network of the following form

$$f(\mathbf{x}) = \mathbf{W}_l \sigma(\mathbf{W}_{l-1} \dots \sigma(\mathbf{W}_1 \mathbf{x})), \quad (10.1)$$

where $\mathbf{W}_i$ has dimension $d_i \times d_{i-1}$, $\mathbf{x} \in \mathbb{R}^{d_0}$, and $\sigma(\cdot)$ is the ReLU activation that is, $\sigma(x) = x \cdot \mathbf{1}_{x \geq 0}$. A second, larger network $g(\mathbf{x})$ is of the following form

$$g(\mathbf{x}) = \mathbf{M}_{2l} \sigma(\mathbf{M}_{2l-1} \dots \sigma(\mathbf{M}_1 \mathbf{x})). \quad (10.2)$$

Our goal is to obtain a pruned version of g by eliminating weights, i.e.,

$$\hat{g}(\mathbf{x}) = (\mathbf{S}_{2l} \odot \mathbf{M}_{2l}) \sigma((\mathbf{S}_{2l-1} \odot \mathbf{M}_{2l-1}) \dots \sigma((\mathbf{S}_1 \odot \mathbf{M}_1) \mathbf{x})), \quad (10.3)$$

where each $\mathbf{S}_i$ is a binary (pruning) matrix, with the same dimension as $\mathbf{M}_i$, and $\odot$ represents element-wise product between matrices. Our objective is to obtain a good approximation while controlling the size of $g(\cdot)$, i.e., the width of $\mathbf{M}_i$'s. In this work we only prune neuron weights, and not entire neurons. We refer the reader to Malach et al. Malach et al. (2020b) for the differences between the two approaches: *pruning weights* and *pruning neurons*.

We consider the case where the weight matrices $\mathbf{M}_{2l}, \dots, \mathbf{M}_1$ of $g(\cdot)$ are randomly

initialized. In particular, each element of the matrices is an independent sample from $U[-1, 1]$.

The error metric that we use is the uniform approximation over the normed-ball, i.e., $\hat{g}$ is ϵ -close to f in the following sense:

$$\max_{\mathbf{x} \in \mathbb{R}^{d_0} : \|\mathbf{x}\| \leq 1} \|f(\mathbf{x}) - \hat{g}(\mathbf{x})\| \leq \epsilon.$$

Observe that a result of this kind can be generalized from the domain $\{\|\mathbf{x}\| \leq 1\}$ to an arbitrarily large radius r , $\{\|\mathbf{x}\| \leq r\}$, by scaling ϵ appropriately. It is necessary, though, to consider only bounded domains because ReLU neural networks are positive-homogeneous ($f(\alpha\mathbf{x}) = \alpha f(\mathbf{x})$ for $\alpha \geq 0$) and thus, any non-zero error can be made arbitrarily large for unbounded domains.

10.3 Lottery Tickets via Subset Sum

We now present our results for approximating a target network by pruning a sufficiently over-parameterized neural network. In fact, we prove that a single random, logarithmically over-parameterized neural network can be pruned to approximate any neural network of a fixed architecture, with high probability. We define $\mathcal{F}$ to be the set of target ReLU neural networks f such that (i) $f : \mathbb{R}^{d_0} \rightarrow \mathbb{R}^{d_l}$, (ii) f has depth l (iii) weight matrix of i -th layer has dimension $d_i \times d_{i-1}$ and spectral norm at most 1. That is,

$$\mathcal{F} = \{f : f(\mathbf{x}) = \mathbf{W}_l \sigma(\mathbf{W}_{l-1} \dots \sigma(\mathbf{W}_1 \mathbf{x})), \forall i \ \mathbf{W}_i \in \mathbb{R}^{d_i \times d_{i-1}} \text{ and } \|\mathbf{W}_i\| \leq 1\} \quad (10.4)$$

We prove that a randomly initialized neural network

$$g(\mathbf{x}) = \mathbf{M}_{2l} \sigma(\mathbf{M}_{2l-1} \dots \sigma(\mathbf{M}_1 \mathbf{x})),$$

which has $2l$ layers and layer widths $\log \frac{d_{i-1}d_i l}{\min\{\epsilon, \delta\}}$ times the corresponding layer widths of $\mathcal{F}$, with probability $1 - \delta$, can approximate any neural network in $\mathcal{F}$ up to error ϵ . For simplicity, we state our results for matrices with spectral norm at most 1, but they can readily be generalized to arbitrary norm bounds.

Theorem 10.3. *Let $\mathcal{F}$ be as defined in Eq. (10.4). Consider a randomly initialized $2l$ -layered neural network*

$$g(\mathbf{x}) = \mathbf{M}_{2l}\sigma(\mathbf{M}_{2l-1} \dots \sigma(\mathbf{M}_1\mathbf{x})),$$

where every weight is drawn from $U[-1, 1]$, $\mathbf{M}_{2i}$ has dimension

$$d_i \times C d_{i-1} \log \frac{d_{i-1}d_i l}{\min\{\epsilon, \delta\}},$$

and $\mathbf{M}_{2i-1}$ has dimension

$$C d_{i-1} \log \frac{d_{i-1}d_i l}{\min\{\epsilon, \delta\}} \times d_{i-1}.$$

Then, with probability at least $1 - \delta$, for every $f \in \mathcal{F}$,

$$\min_{\mathbf{s}_i \in \{0,1\}^{d_i \times d_{i-1}}} \sup_{\|\mathbf{x}\| \leq 1} \|f(\mathbf{x}) - (\mathbf{S}_{2l} \odot \mathbf{M}_{2l})\sigma((\mathbf{S}_{2l-1} \odot \mathbf{M}_{2l-1}) \dots \sigma((\mathbf{S}_1 \odot \mathbf{M}_1)\mathbf{x}))\| < \epsilon.$$

Note that the above result offers a uniform approximation guarantee for all networks in $\mathcal{F}$ by only pruning a single over-parameterized network g . In this sense if $\mathcal{G}$ is the set of all pruned versions of the base neural network $g(\mathbf{x})$, then our guarantee states that, with probability $1 - \delta$,

$$\sup_{f \in \mathcal{F}} \min_{\hat{g}(\mathbf{x}) \in \mathcal{G}} \sup_{\|\mathbf{x}\| \leq 1} \|f(\mathbf{x}) - \hat{g}(\mathbf{x})\| < \epsilon.$$

We note two generalizations of Theorem 10.3 in the remarks below:

Remark 10.4. *Although Theorem 10.3 is stated for initialization with the uniform distribution, Leuker's result allows us to extend it to a wide family of distributions. Suppose P is a univariate*

distribution that contains a uniform distribution in the following sense: there exists a distribution Q , a constant $\alpha \in (0, 1]$, and a constant $c > 0$ such that $P = (1 - \alpha)Q + \alpha U[-c, c]$. Then a random network initialized with P also satisfies the guarantee of Theorem 10.3, up to constants depending on α and c . For example, the standard normal distribution and the Laplace distribution with zero mean and unit variance satisfy this condition.

Remark 10.5. *Theorem 10.3 assumes that both the target network and the over-parameterized network have ReLU activations. This assumption can be relaxed to some extent. Looking at Eq. (F.4) (in the proof of Theorem 10.3, Appendix F.1.3), we see that the target network f can have any activation function, as long as it is 1-Lipschitz. The over-parameterized network g would still have $2l$ layers, where every odd layer would have a ReLU (or linear) activation and every even layer would have the same activation as the target network.*

In Subsection 10.3.1, we illustrate the connection between the SUBSETSUM problem and approximating a single weight via pruning by considering a linear network. Later, in Subsection 10.3.2, we show how to do the same by pruning a ReLU network. Towards the end of Subsection 10.3.2, we outline a proof sketch of Theorem 10.3. The complete proof of Theorem 10.3 can be found in Appendix F.1.

10.3.1 Single Link: Pruning a Linear Network

We now explain our approximation scheme for a single link weight by pruning a random two-layered *linear* neural network. Let the scalar target function be $f(x) = w \cdot x$, where $|w| \leq 0.5$. To make the task simpler, let us assume (just for this subsection) that the second layer is deterministic, and has weights that are all equal to 1. Thus, the over-parameterized neural network $g(\cdot)$, that we will prune, has the following linear architecture:

$$g(x) = \mathbf{1}^T \mathbf{a}x = \sum_{i=1}^n a_i x,$$

where $\mathbf{1}$ is all-ones vector, $\mathbf{a} = [a_1, \dots, a_n]^T$, and the weights a_i are sampled from $U[-1, 1]$. Figure 10.1b shows a visual representation of this network.

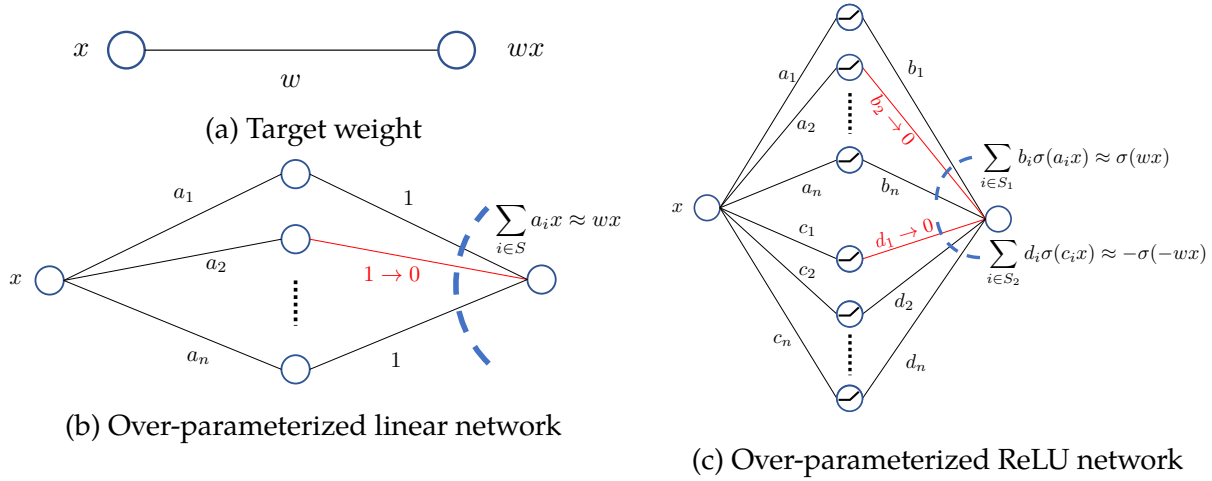


Figure 10.1: (a) Target network is a linear univariate $w x$. (b) A simplified construction, where we approximate the weight w by pruning an over-parameterized two-layered linear network with deterministic second layer of all 1s. We only prune the second layer by setting weights to 0. (c) Our construction from the proof of Lemma 10.7 that uses ReLU units. Our construction adds an additional hidden layer of size $2n = O(\log(1/\epsilon))$. Without loss of generality, assume $w \geq 0$. As the hidden layer also has ReLU activation, we make use of the identity $w x = \sigma(w x) - \sigma(-w x)$ and approximate the two terms separately. We prune the network such that the (i) top half of the network (a_i, b_i) approximates $\sigma(w x)$, (ii) and the bottom half (c_i, d_i) approximates $-\sigma(-w x)$. We only prune the weights in the second layer (shown in red) for a technical reason which helps us reuse the weights in the first layer subsequently in the proof of Theorem 10.3.

Then, the question is how large does n (the width of the random network) need to be so that we can approximate $w x$ by pruning weights in a. As $|x| \leq 1$, we see that it is equivalent to following:

$$\mathbb{P}\left(\exists S \subseteq [n] : \left|w - \sum_{i \in S} a_i\right| \leq \epsilon\right) \geq 1 - \delta, \quad (10.5)$$

where the probability is taken over the randomness in a_i 's. Note that this condition is tightly related to a random instance of the subset sum problem, i.e., $\min_{S \subseteq \{1, \dots, n\}} |w - \sum_{i \in S} a_i|$. This problem was studied by Lueker Lueker (1998b), who obtains the following result:

Theorem 10.6. (Corollary 2.5 Lueker (1998b)) *Let $X_1, \dots, X_n$ be i.i.d. uniform over $[-1, 1]$,*

where $n \geq C \log \frac{2}{\min\{\epsilon, \delta\}}$. Then, with probability at least $1 - \delta$, we have

$$\forall z \in [-0.5, 0.5], \quad \exists S \subset [n] \text{ such that } |z - \sum_{i \in S} X_i| \leq \epsilon.$$

Lueker in Lueker (1998b), establishes this theorem by a beautiful and intricate proof that employs concentration of martingales, and is crucial for the proof of our upper bound. Using Theorem 10.6, we see that if $n \geq C \log(1/\min\{\epsilon, \delta\})$, then Eq. (10.5) holds. We can prune the network by simply setting a_i to 0 for $i \notin S$. Equivalently, we could instead prune the output layer at indices that are not in S , as shown in Figure 10.1b. Note that the dependence on ϵ is only logarithmic, which leads to only logarithmic dependence on width in Theorem 10.3.

10.3.2 Single Link: Pruning a ReLU Network

We now show how the ideas from Subsection 10.3.1 can be extended to random ReLU networks.

Lemma 10.7. (*Approximating a weight*) Let $g : \mathbb{R} \rightarrow \mathbb{R}$ be a randomly initialized network of the form $g(x) = \mathbf{v}^T \sigma(\mathbf{u}x)$, where $\mathbf{v}, \mathbf{u} \in \mathbb{R}^{2n}$, $n \geq C \log \frac{2}{\epsilon}$, and all v_i, u_i 's are drawn i.i.d. from $U[-1, 1]$. Then with probability at least $1 - \epsilon$,

$$\forall w \in [-1, 1], \quad \exists \mathbf{s} \in \{0, 1\}^{2n} : \sup_{x: |x| \leq 1} |wx - (\mathbf{v} \odot \mathbf{s})^T \sigma(\mathbf{u}x)| < \epsilon.$$

Proof. We decompose wx as $wx = \sigma(wx) - \sigma(-wx)$ and WLOG assume $w \geq 0$. Let

$$\mathbf{v} = \begin{bmatrix} \mathbf{b} \\ \mathbf{d} \end{bmatrix}, \mathbf{u} = \begin{bmatrix} \mathbf{a} \\ \mathbf{c} \end{bmatrix}, \mathbf{s} = \begin{bmatrix} \mathbf{s}_1 \\ \mathbf{s}_2 \end{bmatrix},$$

where $\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d} \in \mathbb{R}^n, \mathbf{s}_1, \mathbf{s}_2 \in \{0, 1\}^n$. Thus, $(\mathbf{v} \odot \mathbf{s})^T \sigma(\mathbf{u}x) = (\mathbf{b} \odot \mathbf{s}_1)^T \sigma(\mathbf{a}x) + (\mathbf{d} \odot \mathbf{s}_2)^T \sigma(\mathbf{c}x)$.

See Figure 10.1c for a diagram.

Step 0: Equivalence between pruning $\mathbf{u}$ and $\mathbf{v}$. Note that $(\mathbf{v} \odot \mathbf{s})^T \sigma(\mathbf{u}x) = \mathbf{v}^T \sigma((\mathbf{s} \odot \mathbf{u})x)$. Thus, in the the following construction, we will prune $\mathbf{u}$ (a and c) instead of $\mathbf{v}$, for simplicity.

Step 1: Pre-processing a . Let $\mathbf{a}^+ = \max\{\mathbf{0}, \mathbf{a}\}$ be the vector obtained by pruning all the negative entries of $\mathbf{a}$. Thus, $\mathbf{a}^+$ contains n i.i.d. random variables from the mixture distribution: $(1/2)P_0 + (1/2)U[0, 1]$, where P_0 is the degenerate distribution at 0.

Since $w \geq 0$, then for $x \leq 0$ we have that $\sigma(wx) = \mathbf{b}^T \sigma(\mathbf{a}^+x) = 0$. Moreover, further pruning of $\mathbf{a}^+$ would not affect this equality for $x \leq 0$. We thus focus our attention on $x > 0$ in steps 1 and 2. Therefore, we get that $\sigma(wx) = wx$ and $\mathbf{b}^T \sigma(\mathbf{a}^+x) = \mathbf{b}^T \mathbf{a}^+x = \sum_i b_i a_i^+ x$.

Step 2: Pruning a via SUBSETSUM. Consider the random variable $Z_i = b_i a_i^+$. We show that Theorem 10.6 also holds for Z_i 's (See Corollary F.4 and Corollary F.5 in Appendix F.3). Therefore, as long as $n \geq C \log 2/\epsilon$, with probability $1 - \epsilon/2$, we can choose a subset of $\{Z_1, Z_2, \dots, Z_n\}$ to approximate w up to ϵ . That is with probability $1 - \epsilon/2$,

$$\forall w \in [0, 1], \quad \exists \mathbf{s}_1 \in \{0, 1\}^n : \quad |w - \mathbf{b}^T(\mathbf{s}_1 \odot \mathbf{a}^+)| < \epsilon/2,$$

and because $|x| \leq 1$, we can uniformly bound the error between the linear function wx and the pruned version of the network. Therefore, with probability $1 - \epsilon/2$,

$$\forall w \in [0, 1], \quad \exists \mathbf{s}_1 \in \{0, 1\}^n : \quad \sup_{x \in [-1, 1]} |\sigma(wx) - \mathbf{b}^T \sigma((\mathbf{s}_1 \odot \mathbf{a}^+)x)| < \epsilon/2, \quad (10.6)$$

where we use that for $x < 0$, $\sigma(wx) = \mathbf{b}^T \sigma((\mathbf{s}_1 \odot \mathbf{a}^+)x) = 0$.

Step 3: Pre-processing c . We now turn our attention to $x \leq 0$ with a similar procedure as steps 1 and 2. We begin by pre-processing c . Let $\mathbf{c}^- = \min\{\mathbf{0}, \mathbf{c}\}$ be the vector obtained by pruning the positive entries of $\mathbf{c}$. Therefore, $\mathbf{c}^-$ contains n i.i.d. random variables from the mixture distribution: $(1/2)P_0 + (1/2)U[-1, 0]$, where P_0 is the degenerate distribution at 0.

Step 4: Pruning c via SUBSETSUM. For $x \geq 0$, we have that $\sigma(-wx) = 0$ and $\mathbf{d}^T \sigma(\mathbf{c}^-x) = 0$. Moreover, pruning $\mathbf{c}^-$ further does not affect the equality. Thus, we only consider the

case $x < 0$.

For $x < 0$, we get that $-\sigma(-wx) = -(-wx) = wx$ and also $\sigma(\mathbf{c}^-x) = \mathbf{c}^-x$. Thus $\mathbf{d}^T \sigma(\mathbf{c}^-x) = (\mathbf{d}^T \mathbf{c}^-)x$. Observe that the $b_i a_i^+$ and $d_i c_i^-$ have the exact same distribution and thus similar to the step 2 above, as long as $n \geq C \log 2/\epsilon$, with probability at least $1 - \epsilon/2$,

$$\forall w \in [0, 1], \quad \exists \mathbf{s}_2 \in \{0, 1\}^n : \quad \sup_{x \in [-1, 1]} |-\sigma(-wx) - \mathbf{d}^T \sigma((\mathbf{s}_2 \odot \mathbf{c}^-)x)| < \epsilon/2, \quad (10.7)$$

where we use that $wx = -\sigma(-wx)$ and $\mathbf{d}^T(\mathbf{s}_2 \odot \mathbf{c}^-)x = \mathbf{d}^T \sigma((\mathbf{s}_2 \odot \mathbf{c}^-)x)$ for $x \leq 0$.

Step 5: Tying it all together. Recall that $w \geq 0$. By a union bound, we assume that Equations (10.6) and (10.7) hold with probability at least $1 - \epsilon$. On that event, we get that

$$\begin{aligned} & \inf_{\mathbf{s} \in \{0, 1\}^{2n}} \sup_{|x| \leq 1} |wx - \mathbf{v}^T \sigma((\mathbf{u} \odot \mathbf{s})x)| \\ &= \inf_{\mathbf{s}_1, \mathbf{s}_2} \sup_{|x| \leq 1} |wx - \mathbf{b}^T \sigma((\mathbf{s}_1 \odot \mathbf{a})x) - \mathbf{d}^T \sigma((\mathbf{s}_2 \odot \mathbf{c})x)| \\ &\leq \inf_{\mathbf{s}_1, \mathbf{s}_2} \sup_{|x| \leq 1} |wx - \mathbf{b}^T \sigma((\mathbf{s}_1 \odot \mathbf{a}^+)x) - \mathbf{d}^T \sigma((\mathbf{s}_2 \odot \mathbf{c}^-)x)| \quad (\text{Using steps 1 and 3}) \\ &= \inf_{\mathbf{s}_1, \mathbf{s}_2} \sup_{|x| \leq 1} |\sigma(wx) - \sigma(-wx) - \mathbf{b}^T \sigma((\mathbf{s}_1 \odot \mathbf{a}^+)x) - \mathbf{d}^T \sigma((\mathbf{s}_2 \odot \mathbf{c}^-)x)| \\ &\leq \inf_{\mathbf{s}_1} \sup_{|x| \leq 1} |\sigma(wx) - \mathbf{b}^T \sigma((\mathbf{s}_1 \odot \mathbf{a}^+)x)| + \inf_{\mathbf{s}_2} \sup_{|x| \leq 1} |-\sigma(-wx) - \mathbf{d}^T \sigma((\mathbf{s}_2 \odot \mathbf{c}^-)x)| \\ &\leq \epsilon, \end{aligned}$$

where the last step uses (10.6) and (10.7). □

Proof Sketch of Theorem 10.3 Lemma 10.7 states that we can approximate a single link with a $O(\log(1/\epsilon))$ wide network. Then we show that we can approximate a neuron with d weights using $O(d \log(d/\epsilon))$ wide network. Moreover, reusing weights allows us to approximate a whole layer of d neurons with a $O(d \log(d/\epsilon))$ wide network. Finally, we show that if we can approximate each layer in the target network individually, we also get a good approximation as a whole.

10.4 Lower Bound by Parameter Counting

We now state the lower bound for the required over-parameterization by showing that even approximating a linear network requires blow up of width by $\log(1/\epsilon)$. For a matrix $\mathbf{W}$, we can express the linear function $\mathbf{W}\mathbf{x}$ as a ReLU network $h_{\mathbf{W}}$ (see Eq. (10.8)). Let $\mathcal{F}$ be the set of neural networks, that represent linear functions with spectral norm at most 1, i.e.,

$$\mathcal{F} := \{h_{\mathbf{W}} : \mathbf{W} \in \mathbf{R}^{d \times d} : \|\mathbf{W}\| \leq 1\}, \quad \text{where} \quad h_{\mathbf{W}}(\mathbf{x}) = \begin{bmatrix} \mathbf{I} & -\mathbf{I} \end{bmatrix} \sigma \left(\begin{bmatrix} \mathbf{W} \\ -\mathbf{W} \end{bmatrix} \mathbf{x} \right). \quad (10.8)$$

We prove that if a random network (with arbitrary distribution) approximates every $h_{\mathbf{W}} \in \mathcal{F}$ with probability at least 0.5, then the random network needs at least $d^2 \log(1/\epsilon)$ parameters. For a two-layered network, this means that the width must be at least $d \log(1/\epsilon)$. Note that our lower bound does not require a uniform approximation over $\mathcal{F}$, which is achieved by Theorem 10.3 with $d \log(d/\epsilon)$ width. Therefore, Theorem 10.8 shows a lower bound to the version of the *lottery ticket hypothesis* considered by Malach et al. Malach et al. (2020b), whereas Theorem 10.3 shows an upper bound for a stronger version of the *LTH*. Formally, we have the following theorem:

Theorem 10.8. *Consider a neural network, $g : \mathbb{R}^d \rightarrow \mathbb{R}^d$ of the following form:*

$$g(\mathbf{x}) = \mathbf{M}_l \sigma(\mathbf{M}_{l-1} \dots \sigma(\mathbf{M}_1 \mathbf{x}) \dots),$$

with arbitrary distributions on $\mathbf{M}_1, \dots, \mathbf{M}_l$. Let $\mathcal{G}$ be the set of neural networks that can be formed by pruning g , i.e., $\mathcal{G} := \{\hat{g} : \hat{g}(\mathbf{x}) = (\mathbf{S}_l \odot \mathbf{M}_l) \sigma(\dots \sigma((\mathbf{S}_1 \odot \mathbf{M}_1) \mathbf{x}))\}$, where $\mathbf{S}_1, \dots, \mathbf{S}_l$ are pruning matrices\}. Let $\mathcal{F}$ be as defined in Eq. (10.8). If the following statement holds:

$$\forall h \in \mathcal{F}, \mathbb{P} \left(\exists g' \in \mathcal{G} : \sup_{\mathbf{x} : \|\mathbf{x}\| \leq 1} \|h(\mathbf{x}) - g'(\mathbf{x})\| < \epsilon \right) \geq \frac{1}{2}, \quad (10.9)$$

then $g(\mathbf{x})$ has $\Omega(d^2 \log \frac{1}{\epsilon})$ parameters. Further, if $l = 2$, then width of $g(\mathbf{x})$ is $\Omega(d \log \frac{1}{\epsilon})$.

Remark 10.9. *Note that Theorem 10.8 focuses on approximating (linear) multivariate functions, which is an important building block in approximating multilayer neural networks. At the same time, our bounds are tight only for networks with constant depth. Generalizing Theorem 10.8 for deeper networks would require a better understanding of the increase in representation power of neural networks with depth. We leave further investigation of both of these questions for future work.*

Proof Sketch: Our proof strategy is a counting argument, which shows that $|\mathcal{G}|$ has to be large deterministically. Particularly, there exists a matrix $\mathbf{W}$ that is far from all the pruned networks in $\mathcal{G}$ with high probability, unless $|\mathcal{G}|$ is large enough. Together with the fact that $|\mathcal{G}|$ scales with the number of parameters, we get the desired lower bound. See Appendix F.2 for the complete proof.

10.5 Experiments

We verify our results empirically by approximating a target network via SUBSETSUM in Experiment 1, and by pruning a sufficiently over-parameterized neural network that implements the structures in Figures 10.1b and 10.1c in Experiment 2. In both setups, we benchmark on the MNIST Lecun et al. (1998) dataset, and all training and pruning is accomplished with cosine annealing learning rate decay Loshchilov and Hutter (2016) on a batch size 64 with momentum 0.9 and weight decay 0.0005.

Experiment 1: SUBSETSUM. We approximate a two-layer, 500 hidden node target network with a final test set accuracy of 97.19%. Every weight was approximated in this network with a subset sum of $n = C \log_2(\frac{1}{\epsilon}) \approx 21$ coefficients, for $\epsilon = 0.01$ and $C = 3$. To solve SUBSETSUM more efficiently, we implement the following mixed integer program (MIP) for every weight w in the target network and solve it using Gurobi’s Gurobi Optimization

(2020) MIP solver:

$$\min_{x_1, \dots, x_n} \left| w - \sum_{i=1}^n a_i x_i \right| \quad \text{subject to} \quad \left| w - \sum_{i=1}^n a_i x_i \right| \leq \epsilon, \quad x_i \in \{0, 1\} \quad \forall i = 1, \dots, n, \\ a_i \in [l, u] \quad \forall i = 1, \dots, n,$$

where l and u are the bounds of the uniform coefficient distribution and ϵ is sufficiently large. Every set of a_i coefficients is unique to the approximation of w , and these coefficients are drawn uniformly from a range which is fine-tuned for the target network. We recommend that the set $[l, u]$ be large enough to at least contain all weights in the target network, making the solution to any SUBSETSUM problem less likely to be infeasible. When the bounds are close to the minimum and maximum weight values in the network, we find that we can decrease n , either by increasing ϵ or decreasing C , thereby reducing time complexity. Since the solver finds the optimal solution, most weights in our approximated network were well below 0.01 error, and our approximated network maintained the 97.19% test set accuracy. The 397,000 weights in our target network were approximated with 3,725,871 coefficients in 21.5 hours on 36 cores of a c5.18xlarge AWS EC2 instance. Such a running time is attributed to solving many instances of this nontrivial combinatorial problem.

Experiment 2: Pruning Random Networks. We train baseline networks, including two-layer and four-layer fully connected networks with 500 hidden nodes per layer (learning rate 0.1 for 10 epochs) and LeNet5 (learning rate 0.01 for 50 epochs). In Figure 4, we show the result of implementing the structure in Figure 10.1c (with and without ReLU activation) in each of these networks, and compare the result with their respective baselines and wide-network counterparts. More specifically, we compare the results of pruning our structure with pruning a wider, random network such that the number of parameters is approximately equal to the number of parameters in the network with our structure. We use the edge-popup Ramanujan et al. (2020) algorithm to prune the networks, which finds a subnetwork in each of these two architectures without training the weights. Since

our structure in Figure 10.1c is well-defined only for fully connected layers, we prune LeNet5 by randomly initializing and freezing the convolutional filters and pruning the fully connected layers. The weights and scores in the pruned networks are initialized with a Kaiming Normal He et al. (2015) and Kaiming Uniform distribution, respectively. The LeNet5 networks are pruned with learning rate 0.01; the fully connected networks utilize a learning rate of 0.1. Experiments are run on these pruned architectures for 10, 20, 30, 50, and 100 epochs, and the maximum accuracy for each architecture for a particular number of parameters is then selected. We vary the number of parameters in each network by adjusting the $[0, 1]$ sparsity parameter.

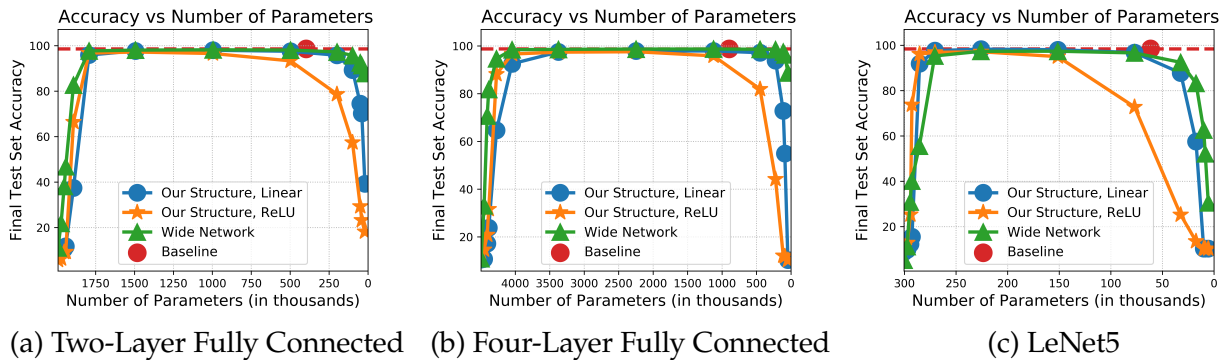


Figure 10.2: Performance of pruning our structure shown in Figure 10.1c, with and without ReLU activation, using 5 (a_i, b_i) coefficient pairs per weight. The number of parameters is a function of the sparsity of each network. Results are benchmarked on MNIST.

Note that the use of additional ReLU activations leads to worse performance than the use of an identity mapping when the number of parameters is small. We posit that the additional sparsification from ReLU degrades the performance of our already sparsified network.

The findings from this second experiment indicate that the performance of a pruning algorithm can vary with regards to its approximation capacity of the target network, depending on the network topology. Note that the different network topologies all contain, approximately, the same number of weights, hence should lead to similar approximations. However we see that the choice of architecture plays an important role in the performance

and efficiency of a given pruning algorithm.

10.6 Discussion

In this paper we establish a tight version of the strong lottery ticket hypothesis: there always exist subnetworks of randomly initialized over-parameterized networks that can come close to the accuracy of a target network; further this can be achieved by random networks that are only a logarithmic factor wider than the original network. Our results are enabled by a very interesting paper on the random subset sum problem from Lueker (1998b), and the essential building block of our analysis is to show that a linear function $f(x) = \sum_{i=1}^d w_i x_i$ on d parameters, can be approximated by selecting a subset of the coefficients of a random one that has $d \log(dl/\epsilon)$ parameters.

Our current work focuses on general fully connected networks. It would be interesting to extend the results to convolutional neural networks. Other interesting structures that come up in neural networks are sparsity and low-rank weight matrices. This leads to the question of whether we can leverage the additional structure in the target network to improve our results. An interesting question from a computational point of view is whether our analysis gives insights to improve the existing pruning algorithms Ramanujan et al. (2020). As remarked in Malach et al. (2020b), the strong LTH implies that pruning an over-parameterized network to obtain good accuracy is NP-Hard in the worst case. It is an interesting future direction to find efficient algorithms for pruning which provably work under mild assumptions on the data.

11 LOGARITHMIC BOUNDS FOR STRONG LTH FOR BINARY NETWORKS

A recent work by Ramanujan et al. (2020) provides significant empirical evidence that sufficiently overparameterized, random neural networks contain untrained subnetworks that achieve state-of-the-art accuracy on several predictive tasks. A follow-up line of theoretical work provides justification of these findings by proving that slightly overparameterized neural networks, with commonly used continuous-valued random initializations can indeed be pruned to approximate any target network. In this work, we show that the amplitude of those random weights does not even matter. We prove that any target network can be approximated up to arbitrary accuracy by simply pruning a random network of binary $\{\pm 1\}$ weights that is only a polylogarithmic factor wider and deeper than the target network.

11.1 Introduction

As the number of parameters of state-of-the-art networks continues to increase, pruning has become a prime choice for sparsifying and compressing a model. A rich and long body of research, dating back to the 80s, shows that one can prune most networks to a tiny fraction of their size, while maintaining high accuracy (Mozer and Smolensky, 1989b; Hassibi and Stork, 1993b; Levin et al., 1994; LeCun et al., 1990; Han et al., 2015b,a; Li et al., 2016; Wen et al., 2016; Hubara et al., 2016, 2017; He et al., 2017; Wu et al., 2016; Zhu et al., 2016; He et al., 2018; Zhu and Gupta, 2017; Cheng et al., 2019; Blalock et al., 2020; Deng et al., 2020).

A downside of most of the classic pruning approaches is that they sparsify a model once it is trained to full accuracy, followed by significant fine-tuning, resulting in a compu-

This chapter presents our paper "Finding nearly everything within random binary networks" (Sreenivasan et al., 2022).

tationally burdensome procedure. Frankle and Carbin (2018a) conjectured the existence of *lottery tickets*, *i.e.*, sparse subnetworks at (or near) initialization, that can be trained—just once—to reach the accuracy of state-of-the-art dense models. This may help alleviate the computational burden of prior approaches, as training is predominantly carried on a much sparser model. The conjectured existence of these lucky tickets is referred to as the Lottery Ticket Hypothesis (LTH). Frankle and Carbin (2018a) and Frankle et al. (2020a) show that not only do *lottery tickets exist*, but also the cost of “*winning the lottery*” is not very high.

Along the LTH literature, a curious phenomenon was observed; even at initialization and in the *complete absence* of training, one can find sub-networks of the random initial model that have prediction accuracy far beyond random guessing (Zhou et al., 2019; Ramanujan et al., 2020; Wang et al., 2019). Ramanujan et al. (2020) reported this in its most striking form: state-of-the-art accuracy models for CIFAR10 and ImageNet, *simply reside* within slightly larger, yet completely random networks, and appropriate pruning—and mere pruning—can reveal them! This “*pruning is all you need*” phenomenon is sometimes referred to as the Strong Lottery Ticket Hypothesis.

A recent line of work attempts to establish the theoretical validity of the Strong LTH by studying the following non-algorithmic question:

Can a random network be pruned to approximate a target function $f(x)$?

Here, f represents a bounded range labeling function that acts on inputs $x \in \mathcal{X}$, and is itself a neural network of finite width and depth. This assumption is not limiting, as neural networks are universal approximators (Stinchcombe, 1989; Barron, 1993a; Scarselli and Tsoi, 1998; Klusowski and Barron, 2018; Perekrestenko et al., 2018; Hanin, 2019; Kidger and Lyons, 2020). Note that the answer to the above question is trivial if one does not constraint the size of the random initial network, for all interesting cases of “*random*”. Indeed, if we start with an exponentially wider random neural network compared to the one representing f , by sheer luck, one can always find weights, for each layer, near-identical to those of any target neural network that is f . Achieving this result with a constrained overparameterization,

i.e., the degree by which the random network to be pruned is wider/deeper than f , is precisely why this question is challenging.

Malach et al. (2020a) were the first to prove that the Strong LTH is true, assuming polynomial-sized overparameterization. Specifically, under some mild assumptions, they showed that to approximate a target network of width d and depth l to within error ϵ , it suffices to prune a random network of width $\tilde{\mathcal{O}}(d^2 l^2 / \epsilon^2)$ and depth $2l$. Pensia et al. (2020) offered an exponentially tighter bound using a connection to the SubsetSum problem. They showed that to approximate a target network within error ϵ , it is sufficient to prune a randomly initialized network of width $\mathcal{O}(d \log(dl/\epsilon))$ and depth $2l$. A corresponding lower bound for constant depth networks was also established. Orseau et al. (2020) were also able to reduce the dependence on ϵ to logarithmic. They show that in order to approximate a target network within error ϵ , it suffices to prune a random network of width $\mathcal{O}(d^2 \log(dl/\epsilon))$ if the weights are initialized with the hyperbolic distribution. However, this bound on overparameterization is still polynomial in the width d .

The above theoretical studies have focused exclusively on continuous distribution for initialization. However, in the experimental work by Ramanujan et al. (2020), the authors manage to obtain the best performance by pruning networks of scaled, *binary weights*. Training binary networks has been studied extensively in the past (Courbariaux et al., 2015; Simons and Lee, 2019) as they are compute, memory and hardware efficient, though in many cases they suffer from significant loss of accuracy. The findings of Ramanujan et al. (2020) suggest that the accuracy loss may not be fundamental to networks of binary weights, when such networks are learned by pruning. Arguably, since “*carving out*” sub-networks of random models is expressive enough to approximate a target function, *e.g.*, according to Pensia et al. (2020); Malach et al. (2020a), one is posed to wonder about the importance of weights altogether. So perhaps, *binary weights is all you need*.

Diffenderfer and Kailkhura (2021) showed that indeed scaled binary networks can be pruned to approximate any target function. The required overparameterization is

Reference	Width	Depth	Total Params	Weights
Malach et al. (2020a)	$\tilde{\mathcal{O}}(\frac{d^2 l^2}{\epsilon^2})$	$2l$	$\tilde{\mathcal{O}}(\frac{d^2 l^3}{\epsilon^2})$	Real
Orseau et al. (2020)	$\mathcal{O}(d^2 \log \frac{dl}{\epsilon})$	$2l$	$\mathcal{O}(d^2 l \log \frac{dl}{\epsilon})$	Real
Pensia et al. (2020)	$\mathcal{O}(d \log \frac{dl}{\min\{\epsilon, \delta\}})$	$2l$	$\mathcal{O}(dl \log \frac{dl}{\min\{\epsilon, \delta\}})$	Real
Diffenderfer and Kailkhura (2021)	$\mathcal{O}(\frac{ld^{1.5}}{\epsilon} + ld \log \frac{ld}{\delta})$	$2l$	$\mathcal{O}(\frac{l^2 d^{1.5}}{\epsilon} + l^2 d \log \frac{ld}{\delta})$	$\{\pm \epsilon\}$
Ours , Theorem 11.3	$\mathcal{O}(d \log^2 \frac{dl}{\epsilon \delta})$	$\mathcal{O}(l \log \frac{dl}{\epsilon})$	$\mathcal{O}(dl \log^3 \frac{dl}{\epsilon \delta})$	$\{\pm 1\}$

Table 11.1: Comparing the upper bounds for the overparameterization needed to approximate a target network (of width d and depth l) within error $\epsilon > 0$ with probability at least $1 - \delta$ by pruning a randomly initialized network. Note that the weights of all the layers in our construction are purely binary $\{\pm 1\}$ except for the last layer which is scaled so that it is $\{\pm \epsilon'\}$ where $\epsilon' = (\epsilon/d^2 l)^l$.

similar to that of Malach et al. (2020a), *i.e.*, polynomial in the width, depth and error of the approximation. Hence in a similar vein to the improvement that Pensia et al. (2020) offered over the bounds of Malach et al. (2020a), we explore whether such an improvement is possible on the results of Diffenderfer and Kailkhura (2021).

Our Contributions: In this work, we offer an exponential improvement to the theoretical bounds by Diffenderfer and Kailkhura (2021), establishing the following.

Theorem 11.1 (informal). *Consider a randomly initialized, FC, binary $\{\pm 1\}$ network of ReLU activations, with depth $\Theta\left(l \log\left(\frac{dl}{\epsilon}\right)\right)$ and width $\Theta\left(d \log^2\left(\frac{dl}{\epsilon \delta}\right)\right)$, with the last layer consisting of scaled binary weights $\{\pm C\}$. Then, there is always a constant C such that this network can be pruned to approximate **any** FC ReLU network, up to error $\epsilon > 0$ with depth l and width d , with probability at least $1 - \delta$.*

Therefore, we show that in order to approximate any target network, it suffices to just prune a logarithmically overparameterized binary network (Figure 11.1). In contrast to Diffenderfer and Kailkhura (2021), our construction only requires that the last layer be scaled while the rest of the network is purely binary $\{\pm 1\}$. We show a comparison of the known Strong LTH results in Table 11.1.

In light of our theoretical results, one may wonder why in the literature of training, *i.e.*, assigning a sign pattern to fixed architecture, binary networks, a loss of accuracy is observed, *e.g.*, Rastegari et al. (2016). Is this an algorithmic artifact, or does pruning

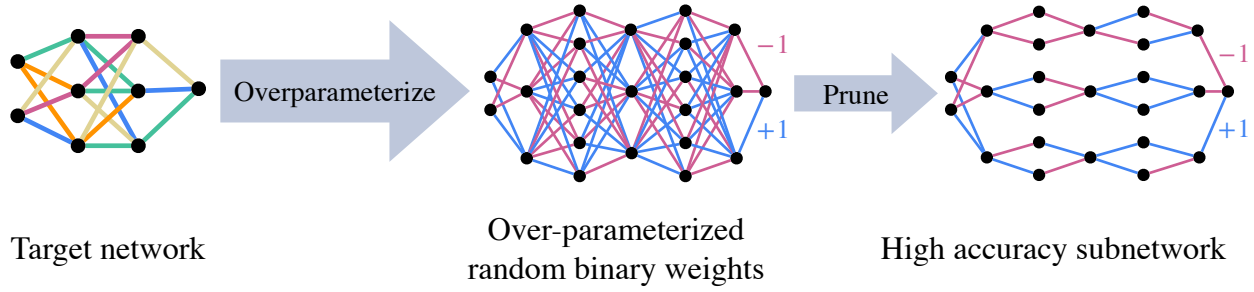


Figure 11.1: Approximating a target network with high accuracy by pruning overparameterized random binary network. In this paper, we show that logarithmic overparameterization in both width and depth is sufficient.

random signs offer higher expressivity than assigning the signs? We show that there exist target functions that can be well approximated by pruning binary networks, yet none of all possible, binary, fully connected networks can approximate it.

Proposition 11.2 (informal). *There exist a function f that can be represented by pruning a random 2-layer binary network of width d , but not by any 2-layer fully-connected binary network of width d .*

Note that although finding a subnetwork of a random binary network results in a “ternary” architecture (e.g., 0 becomes a possible weight), the total number of possible choices of subnetworks is 2^N , if N is the total number of weights. This is equal to the total number of sign assignments of the same FC network. Yet, as shown in the proposition above, pruning a random FC network is provably more expressive than finding a sign assignment for the same architecture.

11.2 Preliminaries and Problem Setup

Let $f(\mathbf{x}) : \mathbb{R}^{d_0} \rightarrow \mathbb{R}$ be the target FC network with l layers and ReLU activations, represented as

$$f(\mathbf{x}) = \sigma(\mathbf{W}_l \sigma(\mathbf{W}_{l-1} \dots \sigma(\mathbf{W}_1 \mathbf{x}))),$$

where $\mathbf{x} \in \mathbb{R}^{d_0}$ is the input, $\sigma(z) = \max\{z, 0\}$ is the ReLU activation and $\mathbf{W}_i \in \mathbb{R}^{d_i \times d_{i-1}}$ is the weight matrix of layer $i \in [l]$. With slight abuse of terminology, we will refer to f as a

network, as opposed to a labeling function. We then consider a binary¹ network of depth l'

$$g(\mathbf{x}) = \sigma((\epsilon' \mathbf{B}_{l'}) \sigma(\mathbf{B}_{l'-1} \dots \sigma(\mathbf{B}_1 \mathbf{x}))),$$

where $\mathbf{B}_i \in \{-1, +1\}^{d'_i \times d'_{i-1}}$ is a binary weight matrix, with all weights drawn uniformly at random from $\{\pm 1\}$, for all layers $i \in [l']$ and the last layer is multiplied by a factor of $\epsilon' > 0$. The scaling factor is calculated precisely in Section 11.3.2.3, where we show that it is unavoidable for function approximation (*i.e.*, regression), rather than classification.

Our goal is to find the smallest network g so that it contains a subnetwork $\tilde{g}$ which approximates f well. More precisely, we will bound the overparameterization of the binary network, under which one can find supermask matrices $\mathbf{M}_i \in \{0, 1\}^{d'_i \times d'_{i-1}}$, for each layer $i \in [l']$, such that the pruned network

$$\tilde{g}(\mathbf{x}) = \sigma(\epsilon' (\mathbf{M}_{l'} \odot \mathbf{B}_{l'}) \sigma((\mathbf{M}_{l'-1} \odot \mathbf{B}_{l'-1}) \dots \sigma((\mathbf{M}_1 \odot \mathbf{B}_1) \mathbf{x})))$$

is ϵ -close to f in the sense of uniform approximation over the unit-ball, *i.e.*,

$$\max_{\mathbf{x} \in \mathbb{R}^{d_0} : \|\mathbf{x}\| \leq 1} \|f(\mathbf{x}) - \tilde{g}(\mathbf{x})\| \leq \epsilon$$

for some desired $\epsilon > 0$. In this paper, we show g only needs to be polylogarithmically larger than the target network f to have this property. We formalize this and provide a proof in the following sections.

Henceforth, we denote $[k] = \{1, 2, \dots, k\}$ for some positive integer k . Unless otherwise specified, $\|\cdot\|$ refers to the ℓ_2 norm. We also use the max norm of a matrix, defined as $\|\mathbf{A}\|_{\max} := \max_{ij} |A_{ij}|$. The element-wise product between two matrices $\mathbf{A}$ and $\mathbf{B}$ is denoted by $\mathbf{A} \odot \mathbf{B}$. We assume without loss of generality that the weights are specified in the base-10 system. However, since we don't specify the base of the logarithm explicitly in our computations, we use the $\Theta(\cdot)$ notation to hide constant factors that may arise from

choosing different bases.

11.3 Strong Lottery Tickets by Binary Expansion

In this section, we formally present our approximation results. We show that in order to approximate any target network $f(\mathbf{x})$ within arbitrary approximation error ϵ , it suffices to prune a random binary¹ network $g(\mathbf{x})$ that is just polylogarithmically deeper and wider than the target network.

11.3.1 Main Result

First, we point out that the scaling factor ϵ' in the final layer of $g(\mathbf{x})$ is necessary for achieving arbitrary small approximation error for any target network $f(\mathbf{x})$. In other words, it is impossible to approximate any arbitrary target network with a purely binary $\{\pm 1\}$ network regardless of the overparameterization. To see this, note that for the simple target function $f(x) = \epsilon x$, $x \in [0, 1]$ and $\epsilon \in [0.5, 1)$, the best approximation possible by a binary network is $g(x) = x$ and therefore $\max_{x \in \mathbb{R}: |x| \leq 1} |f(x) - g(x)| \geq (1 - \epsilon)$ for any binary network g . We will show that just by allowing the weights of the final layer to be scaled, we can provide a uniform approximation guarantee while the rest of the network remains binary $\{\pm 1\}$. Formally, we have the following theorem:

Theorem 11.3. *Consider the set of FC ReLU networks $\mathcal{F}$ defined as*

$$\mathcal{F} = \{f : f(\mathbf{x}) = \sigma(\mathbf{W}_l \sigma(\mathbf{W}_{l-1} \dots \sigma(\mathbf{W}_1 \mathbf{x}))), \forall i \ \mathbf{W}_i \in \mathbb{R}^{d_i \times d_{i-1}} \ ||\mathbf{W}_i|| \leq 1\},$$

and let $d = \max_i d_i$. For arbitrary target approximation error ϵ , let

$g(\mathbf{x}) = \sigma(\epsilon' \mathbf{B}_{l'} \sigma(\mathbf{B}_{l'-1} \dots \sigma(\mathbf{B}_1 \mathbf{x})))$ (here $\epsilon' = (\epsilon/d^2 l)^l$) be a randomly initialized network with depth $l' = \Theta(l \log(d^2 l/\epsilon))$ such that every weight is drawn uniformly from $\{-1, +1\}$ and the layer widths are $\Theta\left(\log d^2 l/\epsilon \cdot \log\left(\frac{dl \log^2(d^2 l/\epsilon)}{\delta}\right)\right)$ times wider than $f(\mathbf{x})$.

Then, with probability at least $1 - \delta$, for every $f \in \mathcal{F}$, there exist pruning matrices $\mathbf{M}_i$ such that

$$\max_{\mathbf{x} \in \mathbb{R}^{d_0}: \|\mathbf{x}\| \leq 1} |f(\mathbf{x}) - \tilde{g}(\mathbf{x})| \leq \epsilon$$

holds where

$$\tilde{g}(\mathbf{x}) := \sigma(\epsilon'(\mathbf{M}_{l'} \odot \mathbf{B}_{l'})\sigma((\mathbf{M}_{l'-1} \odot \mathbf{B}_{l'-1}) \dots \sigma((\mathbf{M}_1 \odot \mathbf{B}_1)\mathbf{x}))).$$

Remark 11.4. The dimensions of the weight matrices of $g(\mathbf{x})$ in Theorem 11.3 are specified more precisely below. Let $p = (d^2 l / \epsilon)$. Since $l' = l \log(p)$, we have $\lfloor \log(p) \rfloor$ layers in $g(\mathbf{x})$ that approximate each layer in $f(\mathbf{x})$. For each $i \in [l]$, the dimension of $\mathbf{B}_{(i-1)\lfloor \log(p) \rfloor + 1}$ is

$$\Theta \left(d_{i-1} \log(p) \log \left(\frac{dl \log^2(p)}{\delta} \right) \right) \times d_{i-1},$$

the dimension of $\mathbf{B}_{i\lfloor \log(p) \rfloor}$ is

$$d_i \times \Theta \left(d_{i-1} \log(p) \log \left(\frac{dl \log^2(p)}{\delta} \right) \right)$$

and the remaining $\mathbf{B}_{(i-1)\lfloor \log(p) \rfloor + k}$ where $1 < k < \lfloor \log(p) \rfloor$ have the dimension

$$\Theta \left(d_{i-1} \log(p) \log \left(\frac{dl \log^2(p)}{\delta} \right) \right) \times \Theta \left(d_{i-1} \log(p) \log \left(\frac{dl \log^2 p}{\delta} \right) \right).$$

11.3.2 Proof of Theorem 11.3

First, we show in Section 11.3.2.1 that any target network in $f(\mathbf{x}) \in \mathcal{F}$ can be approximated within $\epsilon > 0$, by another network $\hat{g}_p(\mathbf{x})$ having weights of finite-precision at most p digits where p is logarithmic in d, l , and ϵ .

Then, in Section 11.3.2.2, we show that any finite precision network can be represented exactly using a binary network where all the weights are binary (± 1) except the last layer, and

the last layer weights are scaled-binary ($\pm\epsilon'$). The proof sketch is as follows. First, through a simple scaling argument we show that any finite-precision network is equivalent to a network with integer weights in every layer except the last. We then present Theorem 11.8 which shows the deterministic construction of a binary network using diamond-shaped gadgets that can be pruned to approximate any integer network. Lemma 11.19 extends the result to the case when the network is initialized with random binary weights.

Putting these together completes the proof of Theorem 11.3 as shown in Section 11.3.2.3.

11.3.2.1 Logarithmic precision is sufficient

First, we consider the simplest setting wherein the target network contains a single weight *i.e.*, $h(x) = \sigma(wx)$, where x, w are scalars, the absolute values of which are bounded by 1. This assumption can be relaxed to any finite norm bound. We begin with noting a fact that $\log(1/\epsilon)$ digits of precision are sufficient to approximate a real number within error ϵ , as formalized below

Fact 11.5. *Let $w \in \mathbb{R}$, $|w| \leq 1$ and $\hat{w}$ be a finite-precision truncation of w with $\lceil \Theta(\log(1/\epsilon)) \rceil$ digits. Then $|w - \hat{w}| \leq \epsilon$ holds.*

Now we state the result for the case when the target network contains a single weight w .

Lemma 11.6. *Consider a network $h(x) = \sigma(wx)$ where $w \in \mathbb{R}, |w| \leq 1$. For a given $\epsilon > 0$, let $\hat{w}$ be a finite-precision truncation of w up to $\log(1/\epsilon)$ digits and let $\hat{g}_{\log(1/\epsilon)}(x) = \sigma(\hat{w}x)$. Then we have*

$$\max_{x \in \mathbb{R}: |x| \leq 1} |h(x) - \hat{g}_{\log(1/\epsilon)}(x)| \leq \epsilon.$$

Proof. By Fact 11.5, we know that $|w - \hat{w}| \leq \epsilon$. Applying Cauchy-Schwarz with $|x| \leq 1$ gives us $|\hat{w}x - wx| \leq \epsilon$. Since this holds for any x and ReLU is 1-Lipschitz, the result follows. $\square$

Lemma 11.6 can be extended to show that it suffices to consider finite-precision truncation up to $\log(d^2 l / \epsilon)$ digits to approximate a network for width d and depth l . This is stated more formally below.

Lemma 11.7. Consider a network $h(\mathbf{x}) = \sigma(\mathbf{W}_l \sigma(\mathbf{W}_{l-1} \dots \sigma(\mathbf{W}_1 \mathbf{x})))$ where $\mathbf{W}_i \in \mathbb{R}^{d_i \times d_{i-1}}$, $\|\mathbf{W}_i\| \leq 1$. For a given $\epsilon > 0$, define $\hat{g}_{\log(d^2 l / \epsilon)}(\mathbf{x}) = \sigma(\widehat{\mathbf{W}}_l \sigma(\widehat{\mathbf{W}}_{l-1} \dots \sigma(\widehat{\mathbf{W}}_1 \mathbf{x})))$ where $\widehat{\mathbf{W}}_i$ is a finite precision truncation of $\mathbf{W}_i$ up to $\log(d^2 l / \epsilon)$ digits, where $d = \max_i d_i$. Then we have

$$\max_{\mathbf{x} \in \mathbb{R}^{d_0} : \|\mathbf{x}\| \leq 1} |h(\mathbf{x}) - \hat{g}_{\log(d^2 l / \epsilon)}(\mathbf{x})| \leq \epsilon.$$

We provide the proof of Lemma 11.7 as well as approximation results for a single neuron and layer in the appendix.

11.3.2.2 Binary weights are sufficient

We begin by showing that any finite-precision FC ReLU network can be represented perfectly as a FC ReLU network with integer weights in every layer except the last, using a simple scaling argument. Since ReLU networks are positive homogenous, we have that $\sigma(c \cdot z) = c \cdot \sigma(z)$ for $c > 0$. Given a network g_p where all the weights are of finite-precision at most p , we can apply this property layerwise with the scaling factor $c = 10^p$ so that,

$$\begin{aligned} f(\mathbf{x}) &= \sigma(\mathbf{W}_l \sigma(\mathbf{W}_{l-1} \dots \sigma(\mathbf{W}_1 \mathbf{x}))) \\ &= \frac{1}{c^l} \sigma(c \mathbf{W}_l \sigma(c \mathbf{W}_{l-1} \dots \sigma(c \mathbf{W}_1 \mathbf{x}))) \\ &= \sigma(c' \widehat{\mathbf{W}}_l \sigma(\widehat{\mathbf{W}}_{l-1} \dots \sigma(\widehat{\mathbf{W}}_1 \mathbf{x}))) \end{aligned} \tag{11.1}$$

where $\widehat{\mathbf{W}}_i = 10^p \mathbf{W}_i$ is a matrix of integer weights and $c' = \frac{1}{c^l}$. Therefore, the rescaled network has integer weights in every layer except the last layer which has the weight matrix $c' \widehat{\mathbf{W}}_l = (c^{-l}) \mathbf{W}_l$.

In the remaining part of this section, we show that any FC ReLU network with integer weights can be represented exactly by pruning a purely binary (± 1) FC ReLU network which is just polylogarithmic wider and deeper. More precisely, we prove the following result.

Theorem 11.8. Consider the set of FC ReLU networks with integer weights $\mathcal{F}_W$ defined as

$$\mathcal{F}_W = \{f : f(\mathbf{x}) = \sigma(\mathbf{W}_l \sigma(\mathbf{W}_{l-1} \dots \sigma(\mathbf{W}_1 \mathbf{x}))), \forall i \mathbf{W}_i \in \mathbb{Z}^{d_i \times d_{i-1}} \|\mathbf{W}_i\|_{\max} \leq W\}$$

where $W > 0$. Define $d = \max_i d_i$ and let $g(\mathbf{x}) = \sigma(\mathbf{B}_{l'} \sigma(\mathbf{B}_{l'-1} \dots \sigma(\mathbf{B}_1 \mathbf{x})))$ be a network with depth $l' = \Theta(l \log(|W|))$ where every weight is uniform-randomly generated from $\{-1, +1\}$ and the layer widths are $\Theta\left(\log |W| \cdot \log\left(\frac{dl \log^2 |W|}{\delta}\right)\right)$ times wider than $f(\mathbf{x})$.

Then, with probability at least $1 - \delta$, for every $f \in \mathcal{F}$, there exist pruning matrices $\mathbf{M}_i$ such that

$$f(\mathbf{x}) = \tilde{g}(\mathbf{x})$$

holds for any $\mathbf{x} \in \mathbb{R}^{d_0}$ where $\tilde{g}(\mathbf{x}) := \sigma((\mathbf{M}_{l'} \odot \mathbf{B}_{l'}) \sigma((\mathbf{M}_{l'-1} \odot \mathbf{B}_{l'-1}) \dots \sigma((\mathbf{M}_1 \odot \mathbf{B}_1) \mathbf{x})))$.

Remark 11.9. The dimensions of the weight matrices of $g(\mathbf{x})$ in Theorem 11.8 are specified more precisely below. Note that we have $\lfloor \log |W| \rfloor$ layers in $g(\mathbf{x})$ that exactly represents each layer in $f(\mathbf{x})$. For each $i \in [l]$, the dimension of $\mathbf{B}_{(i-1)\lfloor \log |W| \rfloor + 1}$ is

$$\Theta\left(d_{i-1} \log |W| \log\left(\frac{dl \log^2 |W|}{\delta}\right)\right) \times d_{i-1},$$

the dimension of $\mathbf{B}_{i\lfloor \log |W| \rfloor}$ is

$$d_i \times \Theta\left(d_{i-1} \log |W| \log\left(\frac{dl \log^2 |W|}{\delta}\right)\right)$$

and the remaining $\mathbf{B}_{(i-1)\lfloor \log |W| \rfloor + k}$ where $1 < k < \lfloor \log |W| \rfloor$ have the dimension

$$\Theta\left(d_{i-1} \log |W| \log\left(\frac{dl \log^2 |W|}{\delta}\right)\right) \times \Theta\left(d_{i-1} \log |W| \log\left(\frac{dl \log^2 |W|}{\delta}\right)\right)$$

Remark 11.10. Note that $\tilde{g}(\mathbf{x})$ is *exactly* equal to $f(\mathbf{x})$. Furthermore, we provide a uniform guarantee for all networks in $\mathcal{F}$ by pruning a single over-parameterized network, like Pensia et al. (2020).

Remark 11.11. *Theorem 11.8 can be made into a deterministic construction for any fixed target network thereby avoiding the $\log(1/\delta)$ overparameterization. We extend to the random initialization setting by resampling the construction a sufficient number of times.*

Remark 11.12. *To resolve issues of numerical overflow, we can insert scaling neurons after every layer.*

Remark 11.13. *The integer assumption can easily be converted to a finite-precision assumption using a simple scaling argument. Since all networks in practice use finite-precision arithmetic, Theorem 11.8 may be of independent interest to the reader. However, we emphasize here that there is no approximation error in this setting. Practitioners who are interested in small error (10^{-k}) can just apply Theorem 11.3 and incur an overparameterization factor of $\mathcal{O}(k)$.*

The proof of Theorem 11.8 will first involve a deterministic construction for a binary network that gives us the desired guarantee. We then extend to the random binary initialization. The construction is based on a diamond-shaped gadget that allows us to approximate a single integer weight by pruning a binary ReLU network with just logarithmic overparameterization.

First, consider a target network that contains just a single integer weight *i.e.*, $h(x) = \sigma(wx)$. We will show that there exists a binary FC ReLU network $g(x)$ which can be pruned to approximate $h(x)$.

Lemma 11.14. *Consider a network $h(x) = \sigma(wx)$ where $w \in \mathbb{Z}$. Then there exists a FC ReLU binary network $g(x)$ of width and depth $\mathcal{O}(\log |w|)$ that can be pruned to $\tilde{g}(x)$ so that $\tilde{g}(x) = h(x)$ for all $x \in \mathbb{R}$.*

Proof. Note that since w is an integer, it can be represented using its binary (base-2) expansion

$$w = \text{sign}(w) \sum_{k=0}^{\lfloor \log_2 |w| \rfloor} z_k \cdot 2^k, \quad z_k \in \{0, 1\}. \quad (11.2)$$

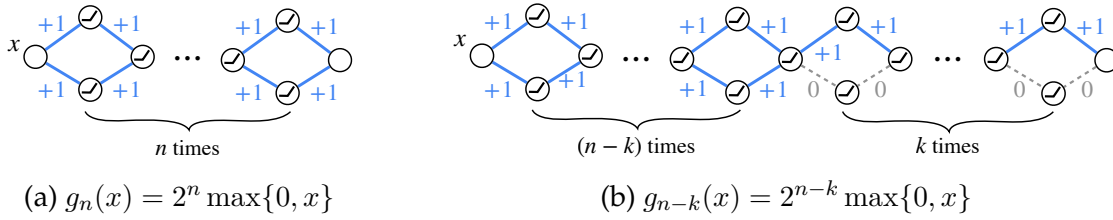


Figure 11.2: The diamond-shaped binary ReLU networks that compute $g_n(x)$ and $g_{n-k}(x)$, respectively. The dashed edges are just weights that have been “pruned” (set to 0). The output neuron is a simple linear activation.

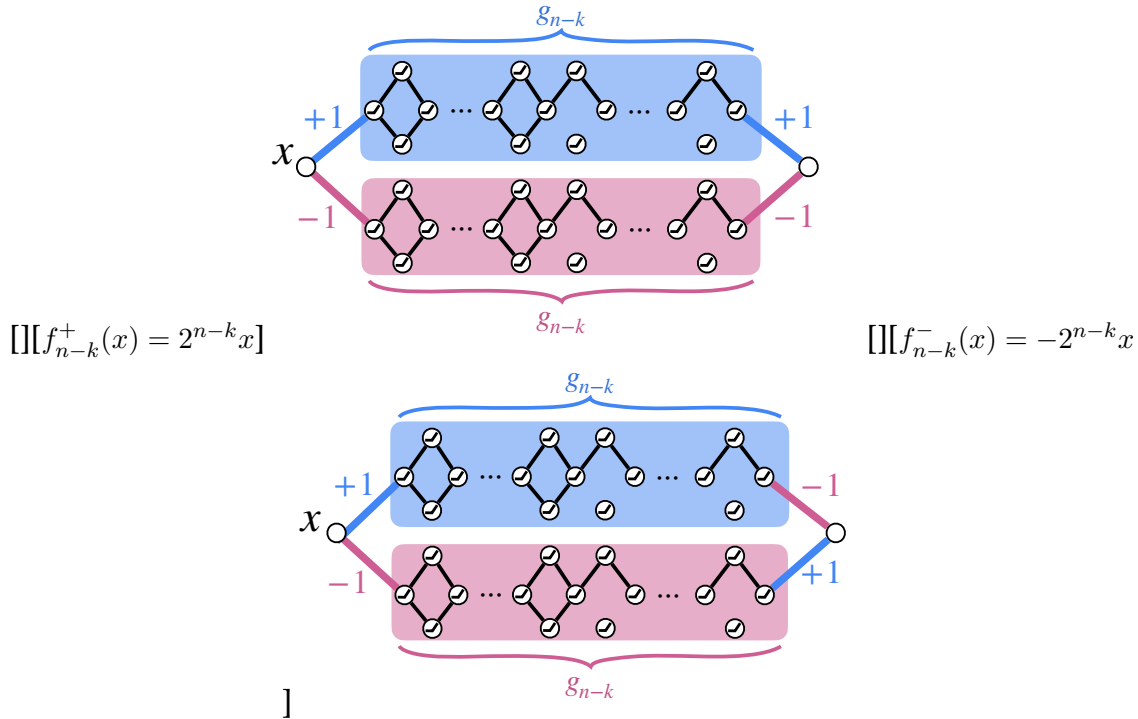


Figure 11.3: We can use two instances of g_{n-k} to create (a) f_{n-k}^+ and (b) f_{n-k}^- to approximate both positive weights and negative weights. The output neuron here has a linear activation.

For ease of notation, we will use $\log(\cdot)$ to represent $\log_2(\cdot)$ going forward. Denote $n = \lfloor \log_2 |w| \rfloor$. The construction of $g_n(x)$ in Figure 11.2a shows that 2^n can be represented by a binary network with ReLU activations for any n . We will refer to this network as the diamond-shaped “gadget”.

Note that the expansion in Equation (11.2) requires 2^k for all $0 \leq k \leq n = \lfloor \log |w| \rfloor$. Luckily, any of these can be represented by just pruning $g_n(x)$ as shown in Figure 11.2b.

However, since our constructions use the ReLU activation, this only works when $x \geq 0$. Using the same trick as Pensia et al. (2020), we can extend this construction by mirroring

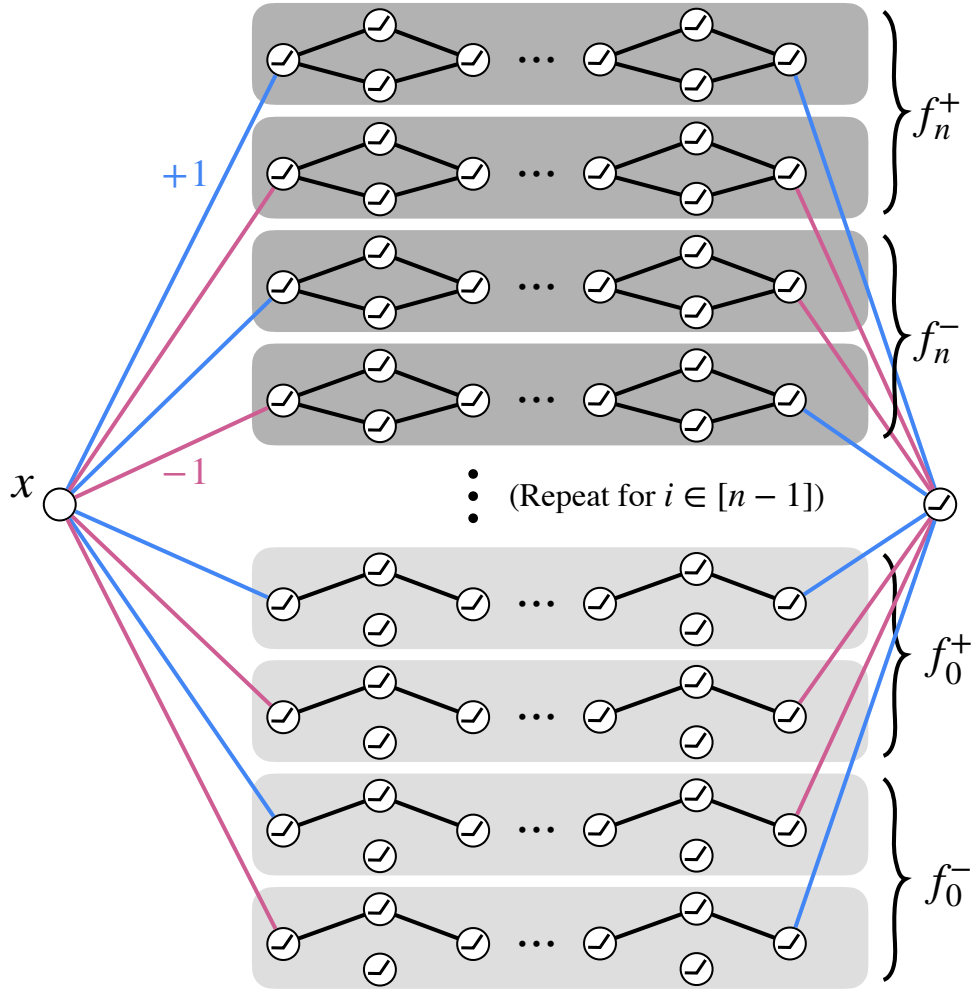


Figure 11.4: Illustration of $g_k(x) = \sum_{k=0}^n (f_k^+(x) + f_k^-(x)) = \sigma(\pm \sum_{k=0}^n 2^k x)$ which can be further pruned to approximate $f(x) = \sigma(wx)$ for any $w : |w| < 2^{n+1} - 1$

it as shown in Figure 11.3. This gives us $f_{n-k}^+(x) := 2^{n-k}x$ and $f_{n-k}^-(x) := -2^{n-k}x$. The correctness of this mirroring trick relies on the simple observation that $wx = \sigma(wx) - \sigma(-wx)$.

Putting these together, we get $g_n(x) = \sigma(\pm \sum_{k=0}^n 2^k x)$ as shown in Figure 11.4. By pruning just the weights in the last layer, we can choose which terms to include. Setting $n = \lfloor \log |w| \rfloor$ completes the approximation.

To calculate the overparameterization required to approximate $h(x) = \sigma(wx)$, we simply count the parameters in the above construction. Each gadget g_k is a network of width 2 and depth $(\lfloor \log |w| \rfloor)$. To construct f_k^+ , we need two such gadgets. Therefore to construct f_k^+ and

f_k^- , we need width 4 and depth $(\lfloor \log |w| \rfloor)$. Repeating this for each $k \in 1, 2, \dots, \lfloor \log |w| \rfloor$ shows that our construction is a network of width and depth $\mathcal{O}(\log |w|)$ which completes the proof of Lemma 11.14. $\square$

Remark 11.15. *The network in Fig. 11.4 used for proving Lemma 11.14 can be written as*

$$g(x) = \sigma(\mathbf{M}_v \odot \mathbf{v})^T [(\mathbf{M}_n \odot \mathbf{B}_n) \sigma((\mathbf{M}_{n-1} \odot \mathbf{B}_{n-1}) \dots \\ \dots \sigma((\mathbf{M}_1 \odot \mathbf{B}_1) \sigma((\mathbf{M}_u \odot \mathbf{u})x)))]],$$

where $\{\mathbf{M}_i\}_{i \in [n]}$, $\mathbf{M}_v$, $\mathbf{M}_u$ are mask matrices and $\{\mathbf{B}_i\}_{i \in [n]}$, $\mathbf{v}$, $\mathbf{u}$ are binary weight matrices. By pruning elements in $\mathbf{u}$ or $\mathbf{v}$, one can obtain $h(x) = \sigma(wx)$. We will always prune the last layer $\mathbf{v}$ as it makes the construction more efficient when we extend it to approximating a layer.

Now, we extend the construction to the case where the target function is a neural network with a single neuron i.e., $h(x) = \sigma(\mathbf{w}^T \mathbf{x})$.

Lemma 11.16. *Consider a network $h(\mathbf{x}) = \sigma(\mathbf{w}^T \mathbf{x})$ where $\mathbf{w} \in \mathbb{Z}^d$ and $\|\mathbf{w}\|_\infty \leq w_{\max}$. Then there exists a FC ReLU binary network $g(\mathbf{x})$ of width $\mathcal{O}(d \log |w_{\max}|)$ and depth $\mathcal{O}(\log |w_{\max}|)$ that can be pruned to $\tilde{g}(\mathbf{x})$ so that $\tilde{g}(\mathbf{x}) = h(\mathbf{x})$ for all $\mathbf{x} \in \mathbb{R}^d$.*

Proof. A neuron can be written as $h(\mathbf{x}) = \sigma(\mathbf{w}^T \mathbf{x}) = \sigma(\sum_{i=1}^d w_i x_i)$. Therefore, we can just repeat our construction from above for each $w_i, i \in [d]$. This results in a network of width $\mathcal{O}(d \log |w_{\max}|)$ while the depth remains unchanged at $\mathcal{O}(\log |w_{\max}|)$. $\square$

Next, we describe how to approximate a single layer target network and avoid a quadratic overparameterization.

Lemma 11.17. *Consider a network $h(\mathbf{x}) = \sigma(\mathbf{1}^T \sigma(\mathbf{W}_1 \mathbf{x}))$ where $\mathbf{W}_1 \in \mathbb{Z}^{d_1 \times d_0}$ and $\|\mathbf{W}_1\|_{\max} \leq W$. Then there exists a FC ReLU binary network $g(\mathbf{x})$ of width $\mathcal{O}(\max\{d_0, d_1\} \log |W|)$ and depth $\mathcal{O}(\log |W|)$ that can be pruned to $\tilde{g}(\mathbf{x})$ so that $\tilde{g}(\mathbf{x}) = h(\mathbf{x})$ for all $\mathbf{x} \in \mathbb{R}^{d_0}$.*

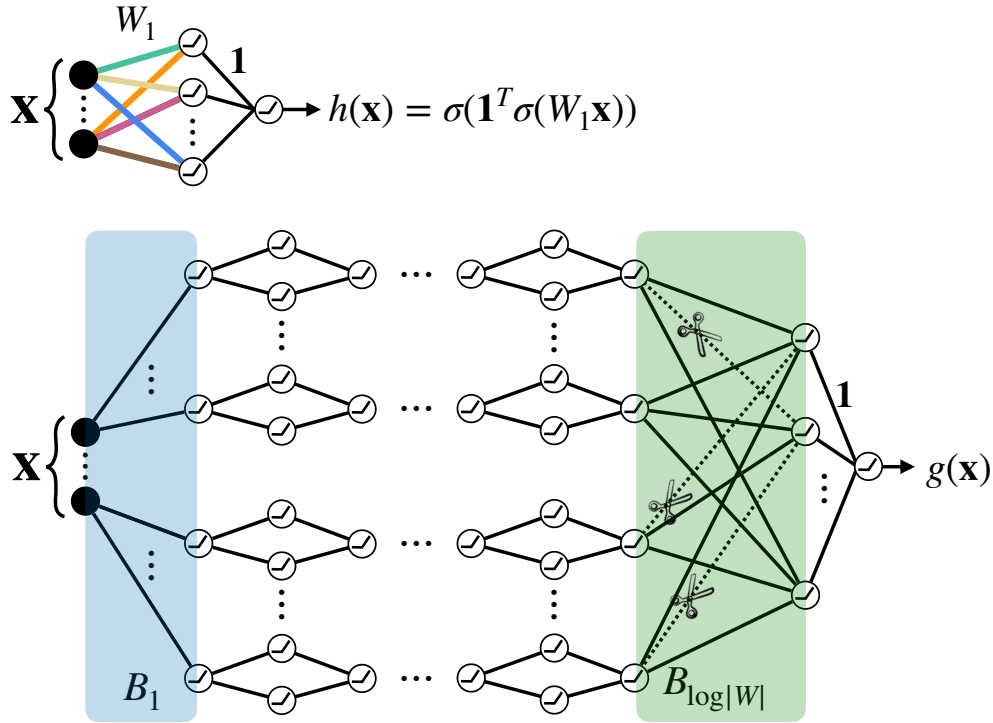


Figure 11.5: Illustration of the construction in Lemma 11.17: Approximating a 1-hidden layer network $h(\mathbf{x}) = \sigma(\mathbf{1}^T \sigma(\mathbf{W}_1 \mathbf{x}))$ by pruning the appropriate binary network $g(\mathbf{x})$. Pruning the last layer allows us to “re-use” weights.

Proof. Note that $\mathbf{1} \in \mathbb{R}^{d_1}$ is the vector of 1’s. Naively extending the construction from Lemma 11.16 would require us to replace each of the d_0 neurons in the first layer by a network of width $\mathcal{O}(d_1 \log |W|)$ and depth $\mathcal{O}(\log |W|)$. This already needs a network of width $\mathcal{O}(d_0 d_1 \log |W|)$ which is a *quadratic* overparameterization. Instead, we take advantage of pruning only the last layer in the construction from Lemma 11.14 to re-use a sufficient number of weights and avoid the quadratic overparameterization. An illustration of this idea is shown in Figure 11.5. The key observation is that by pruning only the last layer of each $f_n(x)$ gadget, we leave it available to be re-used to approximate the weights of the remaining $(d_1 - 1)$ neurons. Therefore, the width of the network required to approximate $h(\mathbf{x})$ is just $\mathcal{O}(\max\{d_0, d_1\} \log |W|)$ and the depth remains $\mathcal{O}(\log |W|)$. $\square$

Now we can tie it all together to show an approximation guarantee for any network in $\mathcal{F}$.

Lemma 11.18. *Consider a network $f(\mathbf{x}) = \sigma(\mathbf{W}_l \sigma(\mathbf{W}_{l-1} \dots \sigma(\mathbf{W}_1 \mathbf{x})))$ where $\mathbf{W}_i \in \mathbb{Z}^{d_i \times d_{i-1}}$, $\|\mathbf{W}_i\|_{\max} \leq W$. Let $d = \max_i d_i$. Then, there exists a FC ReLU binary network $g(\mathbf{x})$ of width $\mathcal{O}(d \log |W|)$ and depth $\mathcal{O}(l \log |W|)$ that can be pruned to $\tilde{g}(\mathbf{x})$ so that $\tilde{g}(\mathbf{x}) = f(\mathbf{x})$ for all $\mathbf{x} \in \mathbb{R}^{d_0}$.*

Proof. Note that there is no approximation error in any of the steps above. Therefore, we can just repeat the construction above for each of the l layers in $f(\mathbf{x})$. This results in a network $g(\mathbf{x})$ of width $\mathcal{O}(d \log |W|)$ and depth $\mathcal{O}(l \log |W|)$. A more precise description of the dimensions of each layer can be found in the statement of Theorem 11.3. $\square$

Finally, below we show that our construction can be extended for networks with randomly initialized weights. The proof can be found in the appendix.

Lemma 11.19. *Consider a network $f(\mathbf{x}) = \sigma(\mathbf{W}_l \sigma(\mathbf{W}_{l-1} \dots \sigma(\mathbf{W}_1 \mathbf{x})))$ where $\mathbf{W}_i \in \mathbb{Z}^{d_i \times d_{i-1}}$, $\|\mathbf{W}_i\|_{\max} \leq W$. Define $d = \max_i d_i$ and let $g(\mathbf{x})$ be a randomly initialized binary network of width $\Theta\left(d \log\left(\frac{(dl \log^2 |W|)}{\delta}\right)\right)$ and depth $\Theta(l \log |W|)$ such that every weight is drawn uniformly from $\{-1, +1\}$. Then $g(\mathbf{x})$ can be pruned to $\tilde{g}(\mathbf{x})$ so that $\tilde{g}(\mathbf{x}) = f(\mathbf{x})$ for all $\mathbf{x} \in \mathbb{R}^{d_0}$ with probability at least $1 - \delta$.*

Since every network $f \in \mathcal{F}_W$ satisfies the assumptions of Lemma 11.19, we can apply it for the entire $\mathcal{F}_W$. Note that we do not need to apply the union bound over $\mathcal{F}_W$ since the randomness is just needed to ensure the existence of a particular deterministic network - specifically $g(\mathbf{x})$ from Lemma 11.18. Therefore, a single random network is sufficient to ensure the guarantee for *every* $f \in \mathcal{F}_W$. This completes the proof of Theorem 11.8.

11.3.2.3 Putting everything together

We now put the above results together to complete the proof of Theorem 11.3. First, note that by Lemma 11.7, to approximate any $f \in \mathcal{F}$ within $\epsilon > 0$, it suffices to consider $\hat{g}(\mathbf{x})$ which is a finite-precision version of f where the precision of each weight is at most $p = \log(d^2 l / \epsilon)$. Now, applying the scaling trick in Equation (11.1) we can represent $\hat{g}(\mathbf{x})$ exactly as a scaled

integer network *i.e.*,

$$\hat{g}(\mathbf{x}) = c^{-l} \sigma(c \widehat{\mathbf{W}}_l \sigma(c \widehat{\mathbf{W}}_{l-1} \dots \sigma(c \widehat{\mathbf{W}}_1 \mathbf{x})))$$

where $c = 10^p = (d^2 l / \epsilon)$ and all the weight matrices $c \widehat{\mathbf{W}}_i$ are integer. Since $\|\mathbf{W}_i\|_{\max} \leq 1$, it is clear that $\|c \widehat{\mathbf{W}}_i\|_{\max} \leq c$. Therefore, applying Theorem 11.8 to the integer network $c^l \hat{g}(\mathbf{x})$ with $W = (d^2 l / \epsilon)$, we have the following. If $h(\mathbf{x}) = \sigma(\mathbf{B}_{l'} \sigma(\mathbf{B}_{l'-1} \dots \sigma(\mathbf{B}_1 \mathbf{x})))$ is a randomly initialized binary network of depth $\Theta(l \log(d^2 l / \epsilon))$ and width $\Theta\left(\log(d^2 l / \epsilon) \log\left(\frac{dl \log^2(d^2 l / \epsilon)}{\delta}\right)\right)$, then with probability $1 - \delta$, it can be pruned to $\tilde{h}(\mathbf{x})$ so that $c^l \hat{g}(\mathbf{x}) = \tilde{h}(\mathbf{x})$ for any $\mathbf{x}$ in the unit sphere. Therefore, to approximate $\hat{g}(\mathbf{x})$ we simply push the scaling factor c^{-l} into the last layer $B_{l'}$ so that its weights are now scaled binary $\{\pm(\epsilon/d^2 l)^l\}$. Combining this with the approximation guarantee between $\hat{g}(\mathbf{x})$ and $f(\mathbf{x})$ completes the proof.

11.3.3 Binary weights for classification

Theorem 11.8 can easily be extended for classification problems using finite precision networks. Since $\text{sign}(\cdot)$ is positive scale-invariant, we no longer even require the final layer to be scaled. Applying the same argument as Sec 11.3.2.3 and then dropping the c^{-l} factor gives us the following corollary.

Corollary 11.20. *Consider the set of binary classification FC ReLU networks $\mathcal{F}$ of width d and depth l , where the weight matrices $\{\mathbf{W}_i\}_{i=1}^l$ are of finite-precision at most p digits. Let $g(\mathbf{x}) = \text{sign}(\mathbf{B}_{l'} \sigma(\mathbf{B}_{l'-1} \dots \sigma(\mathbf{B}_1 \mathbf{x})))$ be a randomly initialized binary network with depth $l' = \Theta(lp)$ and width $d' = \Theta(dp \log(dlp/\delta))$ such that every weight is drawn uniformly from $\{-1, +1\}$. Then, with probability at least $1 - \delta$, for every $f \in \mathcal{F}$, there exist pruning matrices $\{\mathbf{M}_i\}_{i=1}^{l'}$ such that $f(\mathbf{x}) = \tilde{g}(\mathbf{x})$ for any $\mathbf{x}$ where $\tilde{g}(\mathbf{x}) := \text{sign}((\mathbf{M}_{l'} \odot \mathbf{B}_{l'}) \sigma((\mathbf{M}_{l'-1} \odot \mathbf{B}_{l'-1}) \dots \sigma((\mathbf{M}_1 \odot \mathbf{B}_1) \mathbf{x})))$.*

11.4 Conclusion

In this paper, we prove the Strong LTH for binary networks establishing that logarithmic overparameterization is sufficient for pruning algorithms to discover accurate subnetworks within random binary models. By doing this, we provide theory supporting the wide range of experimental work in the field, *e.g.*, scaled binary networks can achieve the best SOTA accuracy on benchmark image datasets (Diffenderfer and Kailkhura, 2021; Ramanujan et al., 2020). Moreover, we show that only the last layer needs to be scaled binary, while the rest of the network can be purely binary $\{\pm 1\}$. It is well known in the binary network literature that a *gain* term (scaling the weights) makes the optimization problem more tractable (Simons and Lee, 2019). While this is known empirically, it would be interesting to study this from a theoretical perspective so we can identify better algorithms to find binary networks of high accuracy.

12 EFFICIENT MEMORIZATION USING THRESHOLD NETWORKS

It is well known that modern deep neural networks are powerful enough to memorize datasets even when the labels have been randomized. Recently, Vershynin (2020) settled a long standing question by Baum (1988), proving that *deep threshold* networks can memorize n points in d dimensions using $\tilde{O}(e^{1/\delta^2} + \sqrt{n})$ neurons and $\tilde{O}(e^{1/\delta^2}(d + \sqrt{n}) + n)$ weights, where δ is the minimum distance between the points. In this work, we improve the dependence on δ from exponential to almost linear, proving that $\tilde{O}(\frac{1}{\delta} + \sqrt{n})$ neurons and $\tilde{O}(\frac{d}{\delta} + n)$ weights are sufficient. Our construction uses Gaussian random weights only in the first layer, while all the subsequent layers use binary or integer weights. We also prove new lower bounds by connecting memorization in neural networks to the purely geometric problem of separating n points on a sphere using hyperplanes.

12.1 Introduction

The current paradigm of training neural networks is to train the networks until they fit the training dataset perfectly (Zhang et al., 2017b; Arpit et al., 2017). This means that the network is able to output the exact label for every sample in the training set, a phenomenon known as *interpolation*. Quite interestingly, modern deep networks have been known to be powerful enough to interpolate even randomized labels (Zhang et al., 2017b; Liu et al., 2020), a phenomenon that is usually referred to as *memorization* (Yun et al., 2019b; Vershynin, 2020; Bubeck et al., 2020), where the networks can interpolate any arbitrary labeling of the dataset.

Given that memorization is a common phenomenon in modern deep learning, a reasonable question to ask is that of how big a neural network needs to be so that it can memorize

This chapter presents our paper "An Exponential Improvement on the Memorization Capacity of Deep Threshold Networks" (Rajput et al., 2021).

a dataset of n points in d dimensions. This question has been of interest since the 80s (Baum, 1988; Mitchison and Durbin, 1989; Sontag, 1990; Huang et al., 1991; Sartori and Antsaklis, 1991). In particular, Baum (1988) proved that a single hidden layer threshold network with $\mathcal{O}(\max(n, d))$ weights and $\mathcal{O}(\lceil n/d \rceil)$ neurons can memorize any set of n points in d dimensions, as long as they are in general position.¹ Baum asked if going deeper could increase the memorization power of threshold networks, and in particular if one could reduce the number of neurons needed for memorization to $\mathcal{O}(\sqrt{n})$. While for ReLU activated networks, Yun et al. (2019b) were able to prove that indeed only $\mathcal{O}(\sqrt{n})$ neurons were sufficient for memorization using deeper networks, the same question remained open for threshold networks.

Recently, Vershynin (2020) was able to answer Baum’s question in the positive by proving that indeed $\tilde{\mathcal{O}}(e^{1/\delta^2} + \sqrt{n})$ neurons were sufficient to memorize a set of n points on a unit sphere separated by a distance at least δ . Many recent works study the memorization power of neural networks under separation assumptions, *e.g.*, Bubeck et al. (2020); Vershynin (2020); Park et al. (2020). One reason why before Vershynin’s work, it was unclear whether going deeper was helpful for threshold networks was that unlike ReLU or sigmoid functions, the threshold activation function, *i.e.*, $\sigma(z) = \mathbb{1}_{z \geq 0}$, prohibits neurons from passing ‘amplitude information’ to the next layer.

Although the dependence of $\mathcal{O}(\sqrt{n})$ on the number of neurons was achieved by the work of Vershynin, it is unclear that the exponential dependence on distance and the requirement that the points lie on a sphere is fundamental. In this work, we lift the spherical requirement and offer an exponential improvement on the dependence on δ .

Theorem 12.1. (*Informal*) *There exists a threshold neural network with $\tilde{\mathcal{O}}\left(\frac{1}{\delta} + \sqrt{n}\right)$ neurons and $\tilde{\mathcal{O}}\left(\frac{d}{\delta} + n\right)$ weights that can memorize any δ -separated dataset of size n in d dimensions.*

Please see Definition 12.4 for the formal definition of δ -separation; informally, it means

¹The “general position” assumption of n vectors in $\mathbb{R}^d$ means that any collection of d of these n vectors are linearly independent.

Reference	# Neurons	# Weights	Assumptions
Baum (1988)	$\mathcal{O}(\lceil n/d \rceil)$	$\mathcal{O}(\max(n, d))$	General Position
Huang et al. (1991)	$\mathcal{O}(n)$	$\mathcal{O}(nd)$	None
Vershynin (2020)	$\mathcal{O}(e^{1/\delta^2} \log^p(n) + \sqrt{n} \log^{2.5}(n))$	$\mathcal{O}(e^{1/\delta^2} (d + \sqrt{n}) + n \log^5(n))$	δ -separation, points lie on the unit sphere
Ours, Theorem 12.8	$\mathcal{O}\left(\frac{\log^2 n}{\delta} + \sqrt{n} \log^2 n\right)$	$\mathcal{O}\left(\frac{(d + \log n) \log n}{\delta} + n \log^2 n\right)$	δ -separation

Table 12.1: Table comparing the upper bounds for the parameters needed for a threshold network to memorize a dataset. General position assumption in $\mathbb{R}^d$ means that no more than d points lie on a $d - 1$ dimensional hyperplane. Note: $\log^p(n)$ denotes a poly-log factor in n .

that all the points have bounded norm and a distance of at least δ between them. Comparing the theorem above with Vershynin’s result, we see that firstly we have reduced the dependence on δ from exponential to nearly linear; and secondly, in our upper bound, the δ and n terms appear in summation rather than product (ignoring the logarithmic factors). Further, note that Vershynin (2020) needs the points to lie on a sphere, whereas we only need them to have a bounded ℓ^2 norm. We have compared our results with the existing work in Table 12.1.

Our construction has $\mathcal{O}(\log \frac{\log n}{\delta})$ layers, of which only the first layer in our construction has real weights, which are i.i.d Gaussian. This layer has $\mathcal{O}(\frac{\log n}{\delta})$ neurons and $\mathcal{O}(\frac{d \log n}{\delta})$ weights. All the other layers have integer weights which are bounded and are on the order $\mathcal{O}(\log n)$.

Baum (1988) also proved that there exists a dataset of n points such that any threshold neural network would need at least $\lceil n/d \rceil$ neurons in the first layer to memorize it. However, the distance for this dataset is $\mathcal{O}(1/n)$.² Our upper bound (Theorem 12.8) shows that if $\delta = \Omega(1/n)$, then we can memorize with only $\mathcal{O}(\frac{\log n}{\delta})$ neurons in the first layer. To complement our upper bound, we introduce a new, δ -dependent lower bound below:

Theorem 12.2. (Informal) *There exists a δ -separated dataset of size $n \in \left[\frac{d^2}{\delta}, \left(\frac{1}{\delta}\right)^{\frac{d}{2}}\right]$ such that any*

²Here, we have rescaled the dataset to have maximum norm of any sample to be 1. This is done to make the dataset consistent with our minimum distance definition (see Assumption 12.6). The assumption in Park et al. (2020) also uses this kind of normalization.

threshold network that can memorize it needs $\tilde{\Omega}\left(\frac{1}{\sqrt{\delta}}\right)$ neurons in the first layer.

The rest of the paper is divided into 7 sections. In Section 12.2, we provide the related works for memorization using neural networks. We provide definitions and notation in Section 12.3 and our main results in Section 12.4. Then, we briefly explain the constructions of our upper bound in Section 12.5. Before exploring lower bounds for threshold networks, we provide sharp bounds on the minimum parameterization needed for any model (not necessarily a neural network) to memorize a δ -separated dataset, in Section 12.6. We discuss our lower bound for threshold networks in Section 12.7. Finally, we conclude our paper with a brief conclusion in Section 12.8.

12.2 Related works

Memorization with threshold activation. There is a rich history of research into the memorization capacity of threshold networks. Baum (1988) showed that a single hidden layer network with $\mathcal{O}(\max(n, d))$ weights and $\mathcal{O}(\lceil n/d \rceil)$ neurons are sufficient to memorize a dataset, *if* the samples are in general positions. Baum also provided an instance of a dataset, where any threshold based network aiming to memorize the dataset would need at least $\lceil n/d \rceil$ neurons in the first layer. However, the construction was such that δ scaled inversely with n . In this work, we show that for any minimum distance $0 < \delta \leq \frac{1}{2}$, the number of neurons in the first layer can be much smaller. Baum also proved, using the Function Counting Theorem (Cover, 1965), that $\mathcal{O}(n/\log n)$ connections are needed in any neural network that aims to memorize points in general position, regardless of the minimum distance.

Huang et al. (1991) and Sartori and Antsaklis (1991) proved that threshold networks could memorize any arbitrary set of points, that is, the points need not be in general position. However, these results needed $(n-1)d$ weights and $n-1$ neurons in the networks for memorization. Mitchison and Durbin (1989) and Kowalczyk (1997) provided upper

and lower bounds when the requirement of memorizing the dataset exactly was relaxed to allow a recall error of up to 50%. Sontag (1990) provided evidence that suggested that adding skip connections can double the memorization capacity of single hidden layer networks. Kotsovsky et al. (2020) extended Baum's result to multiclass classification setting, using bithresholding activation functions.

Recently, Vershynin (2020) was able to show that $\mathcal{O}(e^{1/\delta^2}(d + \sqrt{n}) + n)$ weights and $\mathcal{O}(e^{1/\delta^2} + \sqrt{n})$ neurons are sufficient to memorize a set of n points on the unit sphere, separated by a distance of at least δ . This answered the question by Baum (1988), which was whether the number of neurons could be reduced to $\mathcal{O}(\sqrt{n})$ by considering deeper networks. Note that Vershynin's and Baum's assumptions are fundamentally different and neither is stronger than the other. For example, Baum's construction would still need only $\mathcal{O}(\max(n, d))$ weights even if the points are infinitesimally close to each other, whereas Vershynin's construction would grow as δ decreases. On the other hand, Vershynin's construction works even if all the points lie on a low dimensional hyperplane, whereas Baum's general position assumption would get violated in this situation. Note that naively extending Baum's construction to the case where the points lie on a very low dimensional hyperplane would result in $\mathcal{O}(nd)$ weights and $\mathcal{O}(n)$ neurons, which is identical to the bounds by Huang et al. (1991).

Memorization with ReLU and sigmoid activation. Due to the popularity of the ReLU activation function, there has also been an interest in their memorization capacity. Zhang et al. (2017b) proved that a single hidden layer ReLU network with $2n + d$ weights and n neurons can memorize n arbitrary points. Hardt and Ma (2017) proved that ReLU networks with residual connections also need only $\mathcal{O}(n)$ weights to memorize n points. Yun et al. (2019b) were able to prove that for ReLU, in fact, one needs only $\mathcal{O}(\sqrt{n})$ neurons to memorize n points. Quite interestingly, the construction by Vershynin (2020) works for both threshold activated networks (as described above) as well as ReLU activated networks;

and gives the same bounds for both architectures. Bubeck et al. (2020) show that under some minimum distance assumptions, ReLU networks can memorize (even real labels) with $\mathcal{O}(n/d)$ neurons such that the total weight (Bartlett, 1998) of the network is $\mathcal{O}(\sqrt{n})$, which is proved to be optimal for single hidden layer networks. Huang (2003) proved that a two-hidden layer network with $\mathcal{O}(\sqrt{n})$ sigmoid activated neurons suffices to memorize n points. Recently, Park et al. (2020) have shown that ReLU or sigmoid activated networks with only $\mathcal{O}(n^{2/3} + \log(1/\delta))$ weights are sufficient to memorize n points separated by a normalized distance of δ .

Over parameterization and generalization. A large body of recent work attempts to explain why large neural networks, which are capable memorizers, can also generalize well, *e.g.*, Neyshabur et al. (2018, 2019); Belkin et al. (2020). In this work, we only study memorization, and omit any discussion about generalization. We do not believe our results explain anything about generalization.

12.3 Preliminaries

In this work, we will consider feed forward neural networks $f : \mathbb{R}^d \rightarrow \{0, 1\}$ of the form

$$f(\mathbf{x}) := \sigma(b_L + \mathbf{w}_L^\top \sigma(\mathbf{b}_{L-1} + \mathbf{W}_{L-1} \sigma(\mathbf{b}_{L-2} + \mathbf{W}_{L-2} \sigma(\dots \sigma(\mathbf{b}_1 + \mathbf{W}_1 \mathbf{x}) \dots))),$$

where $\sigma(\cdot)$ is the threshold activation. The network has L layers, with the last layer consisting of a single neuron with bias $b_L \in \mathbb{R}$, and weight vector $\mathbf{w}_L$. For any other layer l , its bias vector is $\mathbf{b}_l$ and weight matrix is $\mathbf{W}_l$. We use the notation $[n]$ to denote the set $\{1, \dots, n\}$.

We formally define memorization as follows:

Definition 12.3. *A learning algorithm $\mathcal{A}$ can memorize a dataset of feature vectors $\mathcal{D} = \{\mathbf{x}_i\}_{i=1}^n$ if for any arbitrary labeling $\{y_i\}_{i=1}^n$ of the dataset, the algorithm can output a model f such that $\forall i \in [n] : f(\mathbf{x}_i) = y_i$.*

For instance, $\mathcal{A}$ could be the training procedure for a particular neural architecture and f could be the neural network that $\mathcal{A}$ outputs after the training process. For neural networks, we will abuse the notation and say that a neural network f can memorize a dataset $\mathcal{D} = \{\mathbf{x}_i\}_{i=1}^n$ if for every arbitrary labeling $\{y_i\}_{i=1}^n$ of the dataset, we can find a set of weights and biases for the network so that $\forall i \in [n] : f(\mathbf{x}_i) = y_i$.

We are mainly interested in the minimum size of a threshold network so that it can memorize a dataset of n samples. Our only assumption on the dataset will be that of δ -separation, which we define below.

Definition 12.4. *We say that a dataset is δ -separated if it satisfies either Assumption 12.5 or Assumption 12.6.*

Assumption 12.5. *(Angular separation) All the feature vectors in the dataset, $\{\mathbf{x}_i\}_{i=1}^n$ satisfy*

$$\arccos \left(\frac{\langle \mathbf{x}_i, \mathbf{x}_j \rangle}{\|\mathbf{x}_i\| \|\mathbf{x}_j\|} \right) \geq \delta$$

Assumption 12.6. *(Normalized minimum distance) All the feature vectors in the dataset, $\{\mathbf{x}_i\}_{i=1}^n$ satisfy $\|\mathbf{x}_i\| \leq 1$ and*

$$\forall i \neq j : \|\mathbf{x}_i - \mathbf{x}_j\| \geq \delta.$$

Remark 12.7. *Note that the ‘normalized’ in the name of Assumption 12.6 refers to the fact that the maximum norm of any point in the dataset is bounded by 1. Our results can be easily extended to any other bounded dataset by rescaling the weights and biases of the first layer appropriately. Park et al. (2020) also use a similar normalized distance assumption.*

Note that if the points lie on the unit sphere, like in Vershynin (2020) or Theorem 12.9 in this paper, then both assumptions are roughly equivalent.

Notation: We use the lower case letters for scalars (w), lower case bold letters for vectors ($\mathbf{w}$), and upper case bold letters for matrices ($\mathbf{W}$). We denote the i -th element of vector $\mathbf{w}$

as w_i , similarly for the case when the vectors have subscript, we denote the i -th element of $\mathbf{w}_j$ by $w_{j,i}$. We use a tilde over $\mathcal{O}(\cdot)$, $\Omega(\cdot)$ and $\Theta(\cdot)$, i.e., $\tilde{\mathcal{O}}(\cdot)$, $\tilde{\Omega}(\cdot)$ and $\tilde{\Theta}(\cdot)$, to hide the logarithmic factors.

12.4 Main results

As discussed earlier, the existing result by Vershynin (2020) has an exponential dependence on δ . In this section, we provide a new and tighter upper bound on the number of weights and neurons needed for a memorization, which brings the dependence down to almost linear. Our result is stated below:

Theorem 12.8. *There exists a threshold activated neural network with $\mathcal{O}(\frac{\log^2 n}{\delta} + \sqrt{n} \log^2 n)$ neurons and*

$\mathcal{O}\left(\frac{(d+\log n) \log n}{\delta} + n \log^2 n\right)$ weights that can memorize any dataset of size n in d dimensions that is δ -separated.

In this construction, only the first layer has real weights. The rest of the network has binary and integer weights. In particular, the $\mathcal{O}(n \log^2 n)$ term in the number of weights comes from binary and integer weights (with the integers being bounded and on the order of $\mathcal{O}(\log n)$). Hence, the $\mathcal{O}(n \log^2 n)$ weights can actually be represented by $\mathcal{O}(n \log^2 n \log \log n)$ bits. We will see in Section 12.6 that $\Omega(n)$ bits are necessary for memorization by any model and hence this term is optimal, up to logarithmic factors. The $\frac{d \log n}{\delta}$ weights in the bound are the weights of the first layer which has width $\frac{\log n}{\delta}$, for which later, in Theorem 12.9, we will provide some indication that this factor might also be necessary.

Baum (1988) had constructed a dataset with normalized minimum distance $\mathcal{O}(1/n)$ and proved that for this dataset, any threshold network aiming to memorize this must have $\lceil n/d \rceil$ neurons. We introduce a new, δ -dependent lower bound below:

Theorem 12.9. *For every $0 < \delta \leq \frac{1}{2}$ and some universal constants C_1 and C_2 , there exists a dataset of size $n \in \left[\frac{C_1 d^2 \log^2(d/\delta)}{\delta}, \left(\frac{C_2}{\delta} \right)^{\frac{d}{2}} \right]$ satisfying the δ -separation condition such that any threshold network that can memorize it needs $\Omega\left(\frac{1}{\sqrt{\delta} \log(1/\delta)}\right)$ neurons in the first layer.*

Note that the dataset in this theorem actually satisfies *both* Assumption 12.5 and Assumption 12.6, up to a constant factor. To understand the implication of Theorem 12.9 a bit better, consider two dataset $\mathcal{D}_1$ and $\mathcal{D}_2$, both containing n points on the sphere. We assume that $\mathcal{D}_1$ has δ_1 -separation and $\mathcal{D}_2$ has δ_2 -separation, where $\delta_1 = \Theta(n^{-2/d})$ and $\delta_2 = \tilde{\Theta}(d^2/n)$. Then, Theorem 12.8 says that there exists a network that can memorize $\mathcal{D}_1$ with only $\mathcal{O}(n^{2/d} \log n)$ neurons in the first layer, whereas Theorem 12.9 says that any network aiming to memorize $\mathcal{D}_2$ would need at least $\tilde{\Omega}(\sqrt{n}/d)$ neurons in the first layer. Hence, we see that the minimum distance can have a big impact on the network architecture.

12.5 Memorization with threshold networks

In this section, we give a proof sketch of Theorem 12.8, the complete proof is provided in the appendix. We will construct a network that can memorize any δ -separated dataset. As we discussed before, the threshold activation prohibits the passing of amplitude information to deeper layers. This is the biggest obstacle in leveraging the benefits of multiple layers of transformation that can be obtained by networks with activations such as ReLU. In the following, we will explain our construction layer-wise:

Step 1: Generating unique binary representations. Given that the amplitude information is lost after thresholding, we want to ensure that at the least, the first layer is able to transform the inputs into binary representations such that each sample in our dataset has a unique binary representation. Later, we will see that this is sufficient for memorization.

Now we give an overview of how a layer with $\mathcal{O}(\frac{\log n}{\delta})$ neurons with random i.i.d. weights can convert the input into unique binary vectors of length $\mathcal{O}(\frac{\log n}{\delta})$. In this sketch,

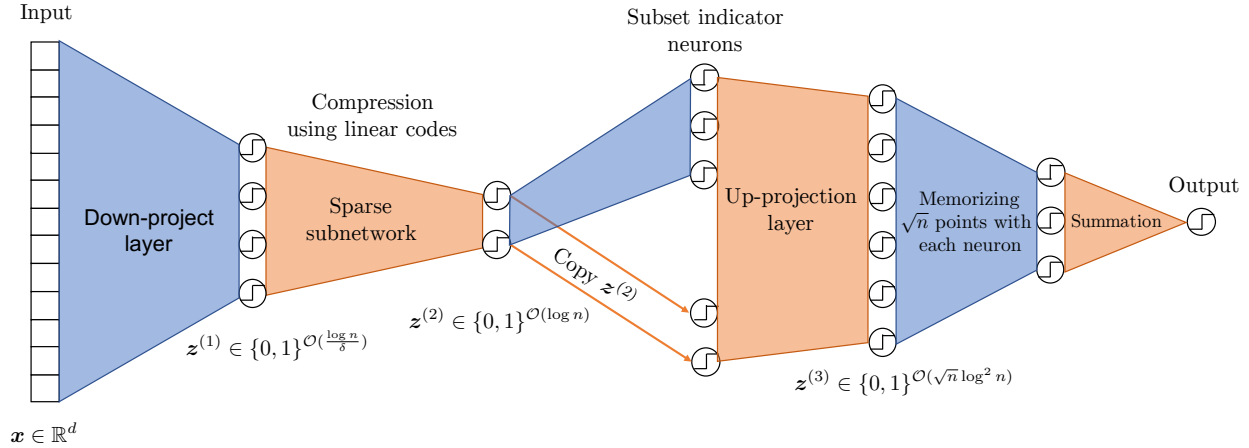


Figure 12.1: The schematic of the neural network that can memorize any δ -separated dataset of size n in d dimensions (Theorem 12.8). The first layer has random Gaussian weights and projects the input from $\mathbb{R}^d$ down to $\{0, 1\}^{\mathcal{O}(\frac{\log n}{\delta})}$. The next component is a sparse subnetwork that further compresses the vectors from $\{0, 1\}^{\mathcal{O}(\frac{\log n}{\delta})}$ to $\{0, 1\}^{\mathcal{O}(\log n)}$ using linear codes. The next couple of layers lift the vectors from $\{0, 1\}^{\mathcal{O}(\log n)}$ to $\{0, 1\}^{\mathcal{O}(\sqrt{n} \log^2 n)}$ while ensuring that we can find sets of vectors of size $\sqrt{n}$ that are linearly independent. Finally, we use this linear independence to memorize up to $\sqrt{n} \log^2 n$ points with a single neuron. This gives us a layer of size $\sqrt{n}$ that can memorize the entire dataset. The final layer is just a single neuron that sums the outputs of these $\sqrt{n}$ neurons.

we will do this under Assumption 12.5, but the same technique can be extended for Assumption 12.6.

The key result we use is that any hyperplane passing through the origin, with random Gaussian coefficients has at least a $\delta/2\pi$ probability of separating two points which have an angle of at least δ between them. To see how this is true, consider two points x_i and x_j with an angle δ between them. Consider the 2-dimensional space spanned by these two vectors. Note that the intersection of the Gaussian hyperplane with the 2-dimensional space spanned by x_i and x_j is just a line passing through the origin. Further, because the Gaussian hyperplane has isometric coefficients, it can be shown that the line has angle uniformly distributed in $[0, 2\pi)$. This implies that the probability that this line passes in between the two points is at least $\delta/2\pi$. If we take m such independent Gaussian hyperplanes, then the probability that none of them separate a pair x_i, x_j is less than $(1 - \frac{\delta}{2\pi})^m$. Finally, taking a simple union bound over the $\binom{n}{2}$ pair of points, shows that the probability that at least one

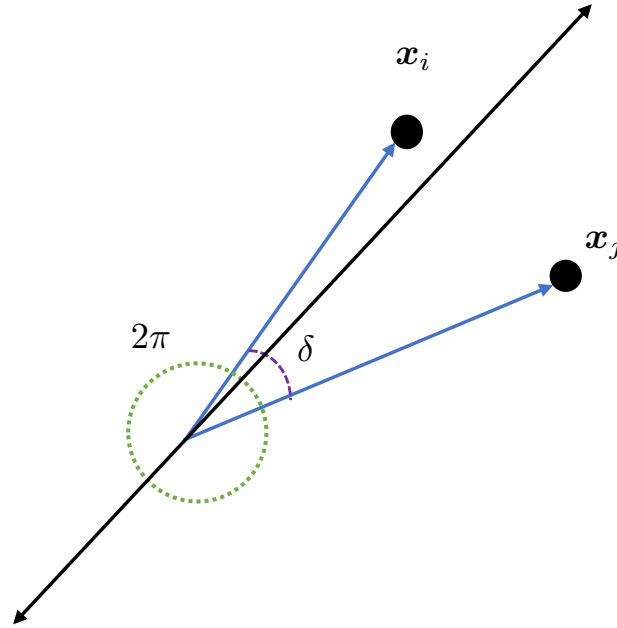


Figure 12.2: The probability that a uniformly random line passing through the origin passes in between x_i and x_j is $\delta/2\pi$.

pair has no hyperplane passing in between them is at most

$$\binom{n}{2} \left(1 - \frac{\delta}{2\pi}\right)^m.$$

We want this to be less than 1 to show that there exists one set of hyperplanes which can separate all the pairs. Doing that gives us $m = \mathcal{O}(\frac{\log n}{\delta})$, which is our required bound.

For any input x_i , each threshold neuron in the first layer outputs either 0 or 1. We can concatenate the outputs of all the neurons in the first layer into a binary vector. Let us denote this binary representation of x_i created by the first layer as $z_i^{(1)}$, for $i = 1, \dots, n$. Further, denote the dimension of $z_i^{(1)}$ by $d^{(1)}$, which is equal to the number neurons in the first layer, that is, $d^{(1)} = \mathcal{O}(\frac{\log n}{\delta})$.

Step 2: Compressing the binary representations to optimal length. In order to avoid a factor δ in the later layers, our next task is to further compress the $\mathcal{O}(\frac{\log n}{\delta})$ dimensional vectors returned by the first layer into vectors of length $\mathcal{O}(\log n)$. Note that $\mathcal{O}(\log n)$ length

is optimal (up to constant multiplicative factors) if we want to represent the n samples uniquely using binary vectors. For this we will use linear codes where we simulate XOR operations by threshold activations.

Consider two $d^{(1)}$ -dimensional binary vectors $\mathbf{z}_i^{(1)}$ and $\mathbf{z}_j^{(1)}$ that differ in at least one bit. Let $\oplus$ represent the XOR operation. Let $\mathbf{b}$ be a random binary vector of length $d^{(1)}$ where each bit is i.i.d Bernoulli(0.5). Then, we claim that

$$\Pr(\langle \mathbf{z}_i^{(1)}, \mathbf{b} \rangle_{\oplus} \neq \langle \mathbf{z}_j^{(1)}, \mathbf{b} \rangle_{\oplus}) = 0.5,$$

$$\text{where } \langle \mathbf{z}, \mathbf{b} \rangle_{\oplus} = (z_1 \cdot b_1) \oplus \cdots \oplus (z_{d^{(1)}} \cdot b_{d^{(1)}})$$

For the ease of exposition, assume that $\mathbf{z}_i^{(1)}$ and $\mathbf{z}_j^{(1)}$ differ in only the first bit (the proof of the general case is provided in the appendix). Let the first bit of $\mathbf{z}_i^{(1)}$ be 0, and that of $\mathbf{z}_j^{(1)}$ be 1, and we have assumed that the rest of their bits are the same. Let $c := (z_{i,2}^{(1)} \cdot b_2) \oplus \cdots \oplus (z_{i,d^{(1)}}^{(1)} \cdot b_{d^{(1)}}) = (z_{j,2}^{(1)} \cdot b_2) \oplus \cdots \oplus (z_{j,d^{(1)}}^{(1)} \cdot b_{d^{(1)}})$. Then,

$$\Pr(\langle \mathbf{z}_i^{(1)}, \mathbf{b} \rangle_{\oplus} \neq \langle \mathbf{z}_j^{(1)}, \mathbf{b} \rangle_{\oplus}) = \Pr(((0 \cdot b_1) \oplus c) \neq ((1 \cdot b_1) \oplus c)).$$

It can now be verified that if $b_1 = 1$, then $((0 \cdot b_1) \oplus c) \neq ((1 \cdot b_1) \oplus c)$. Since $\Pr(b_1 = 1) = 0.5$, we get the desired result.

We have shown that the probability that one such random vector $\mathbf{b}$ differentiates between a pair $\mathbf{z}_i^{(1)}$ and $\mathbf{z}_j^{(1)}$ with probability 0.5. Doing an analysis similar to the one that we did for the first layer, we get that $m = \mathcal{O}(\log n)$ vectors suffice to separate all pairs. The big task here, however, is to implement the XOR operation using threshold activations. In the appendix, we show that this can in fact be achieved using a couple of sparse layers.

Similar to the output of the first layer, let us denote the compressed binary representation of $\mathbf{z}_i^{(1)}$ created by these sparse layers as $\mathbf{z}_i^{(2)}$, for $i = 1, \dots, n$. Further, let $\mathbf{z}_i^{(2)}$ have dimension $d^{(2)}$, where $d^{(2)} = \mathcal{O}(\log n)$, as we showed above.

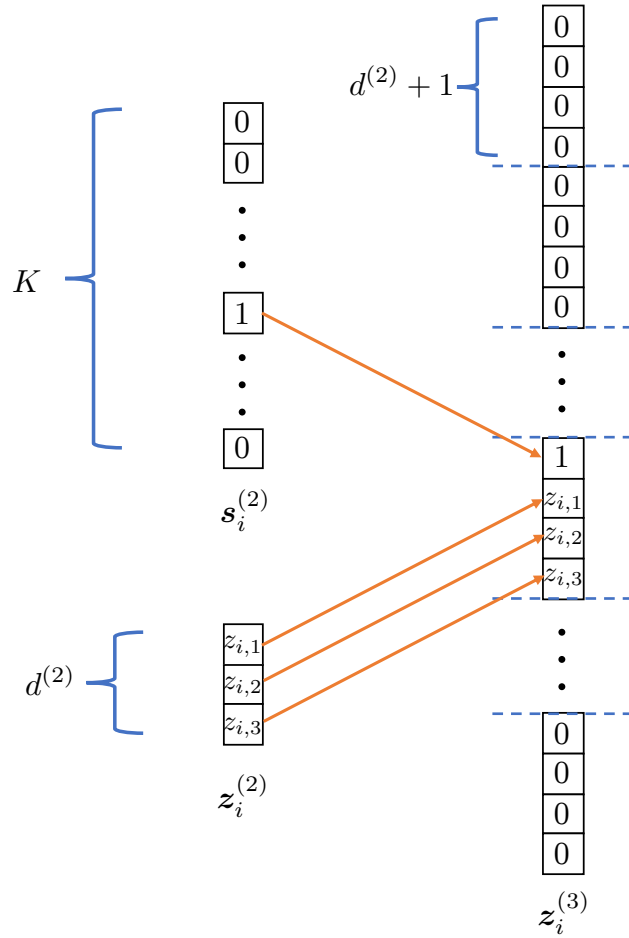


Figure 12.3: In Step 3, $z_i^{(2)}$ is copied to a location in $z_i^{(3)}$ depending on the subset S_j to which $z_i^{(2)}$ belongs.

Step 3: Partitioning the samples into subsets of size $\mathcal{O}(\sqrt{n} \log n)$. Once we have $d^{(2)} = \mathcal{O}(\log n)$ length binary representations of the inputs, memorizing with a single layer consisting of $(d^{(2)} + 1)n$ weights and n neurons is not difficult. However, our aim is to reduce the number of neurons to $\mathcal{O}(\sqrt{n} \log n)$. For that, we use a strategy similar to Yun et al. (2019b): we will memorize up to K samples with 1 neuron, where $K = \Omega(\sqrt{n} \log n)$. Thus, we will need only $\mathcal{O}(\sqrt{n})$ neurons in total.

Roughly speaking the strategy would be the following: Let's say we have a neuron $\sigma(\mathbf{w}^\top \mathbf{z} + b)$. Then, if for any $\{y_1, \dots, y_K\} \in \{0, 1\}^K$, we can find a $\mathbf{w}$ and b such that all of

the following equations hold at the same time,

$$\mathbf{w}^\top \mathbf{z}_1 + b = y_1, \quad \dots, \quad \mathbf{w}^\top \mathbf{z}_K + b = y_K, \quad (12.1)$$

then this neuron can memorize the K samples $\{\mathbf{z}_1, \dots, \mathbf{z}_K\}$.

To do so, we need that the vectors $\{\mathbf{z}_1, \dots, \mathbf{z}_K\}$ be linearly independent. Since we have $K = \Omega(\sqrt{n} \log n)$, we require that $\mathbf{z}_i$'s have dimension at least K . However, the previous few layers have compressed the samples into $d^{(2)} = \mathcal{O}(\log n)$ length binary vectors.

Thus, the next task is to project these $d^{(2)}$ length binary vectors to $\Omega(K)$ length binary vectors. Note that first compressing down to $\mathcal{O}(\log n)$ length vectors and then expanding these up to $\Omega(\sqrt{n} \log n)$ length binary vectors seems counter intuitive. However, we were unable to project the vectors directly to $\Omega(\sqrt{n} \log n)$ length binary vectors without incurring a large dependence on δ and n in the number of weights and neurons.

One more objective that we accomplish while projecting up to $\Omega(K)$ dimension is that both the projection and memorizing the resulting vectors can be easily done with bounded integer weights. The way we do this is by partitioning the n vectors $\mathbf{z}_i^{(2)}$ into K subsets of size at most $\sqrt{n}$ each. We prove that we can find such K subsets, such that each is characterized by a unique prefix, which all the elements of that subset share. These prefixes ensure that we can easily detect which subset a particular $\mathbf{z}_i^{(2)}$ belongs to, using threshold neurons. For more details on this, please refer to the appendix.

Let $\{S_1, \dots, S_K\}$ denote the K subsets, each with a unique prefix. Let $\mathbf{z}_i^{(2)} \in \{0, 1\}^{d^{(2)}}$ be the binary vector returned by **Step 2** above for input $\mathbf{x}_i$. We mentioned above that by using threshold neurons, we can create a vector $\mathbf{vs}_i^{(2)} \in \{0, 1\}^{d^{(2)}}$ for $\mathbf{z}_i^{(2)}$ such that $s_{i,j} = 1$ if and only if $\mathbf{z}_i^{(2)} \in S_j$. Then, using these prefixes and threshold neurons, we can create the expanded binary vector $\mathbf{z}_i^{(3)} \in \{0, 1\}^{K(d^{(2)}+1)}$ corresponding to the vector $\mathbf{z}_i^{(2)}$. The way we create $\mathbf{z}_i^{(3)}$ from $\mathbf{z}_i^{(2)}$ is as follows: $\mathbf{z}_i^{(3)}$ is partitioned into K chunks of length $d^{(2)} + 1$. If $\mathbf{z}_i^{(2)} \in S_j$, then the first bit of the j -th chunk is set to 1 and $\mathbf{z}_i^{(2)}$ is copied to the rest of the bits

in that chunk. All the other bits in $z_i^{(3)}$ are 0. This is shown in Figure 12.3. This operation can be done with threshold activated neuron, for details please refer to the appendix.

Step 4: Memorizing the samples with $\sqrt{n}$ neurons A key property about the vectors $z_i^{(3)}$ is that if we take K such vectors, each belonging to a different subset S_j , then they will be linearly independent as their coordinates do not overlap. Thus, we can find w and b such that (12.1) is satisfied. Furthermore, due to the fact that these K vectors have different supports, we can manage to satisfy (12.1) with w which have integer elements on the order of $\mathcal{O}(\log n)$.

Hence, we can memorize up to K samples with one neuron. This gives that we need $\sqrt{n}$ in the second-to-last layer to memorize all the samples. The final layer is simply a single neuron that sums the outputs of the second-to-last layer.

12.6 Bit complexity of memorization

Before understanding the minimum size that threshold networks have to be in order to memorize a dataset, a more fundamental question is how many bits are needed to specify the models output by any learning algorithm that aims to memorize a dataset. In this section, we provide two theorems that sharply characterize the bit complexity needed for memorization. The results provide the bounds in the terms of two very related quantities, the *packing number* and the *covering number*, which we define below.

Definition 12.10. *Given a real number $\delta > 0$, the packing number $\mathcal{P}_\delta$ of a set S (equipped with norm $\|\cdot\|$) is defined as the cardinality of the largest set $\mathcal{X} \subset S$ such that*

$$\forall x_1, x_2 \in \mathcal{X} : \|x_1 - x_2\| \geq \delta.$$

Intuitively, the packing number is the maximum number of $\delta/2$ radius non-overlapping balls that one can fit inside a set.

Definition 12.11. Given a real number $\delta > 0$, the covering number $\mathcal{C}_\delta$ of a set $\mathcal{S}$ (equipped with norm $\|\cdot\|$) is defined as the cardinality of the smallest set $\mathcal{X} \subset \mathcal{S}$ such that

$$\forall x \in \mathcal{S}, \exists \hat{x} \in \mathcal{X} : \|x - \hat{x}\| \leq \delta.$$

Intuitively, the covering number is the minimum number of δ radius (possibly overlapping) balls that one needs to completely cover a set.

These quantities are very useful in quantifying the ‘size’ of a set. These are related to each other through the following relation (Vershynin, 2018a, Lemma 4.2.8):

$$\mathcal{P}_{2\delta} \leq \mathcal{C}_\delta \leq \mathcal{P}_\delta.$$

Now, we are ready to state the upper and lower bounds on the bit complexity of memorization.

Theorem 12.12. Let $\mathcal{S}$ be a set from which we select the samples. Let $\mathcal{A}$ be a learning algorithm that can memorize any dataset of feature vectors $\mathcal{D} \in \mathcal{S}^n$ as long as every pair of feature vectors $\mathbf{x}_i, \mathbf{x}_j \in \mathcal{D}$ satisfies $\|\mathbf{x}_i - \mathbf{x}_j\| \geq \delta$. Then, we need at least $\max(n, \log_2 \log_2 \mathcal{P}_\delta)$ bits to represent the models output by $\mathcal{A}$.

Theorem 12.13. Let $\mathcal{S}$ be a set from which we select the samples. Then, there exists a learning algorithm $\mathcal{A}$ that can memorize any dataset of feature vectors $\mathcal{D} \in \mathcal{S}^n$ as long as every pair of feature vectors $\mathbf{x}_i, \mathbf{x}_j \in \mathcal{D}$ satisfies $\|\mathbf{x}_i - \mathbf{x}_j\| \geq \delta$, and the models returned by $\mathcal{A}$ can be represented in $O(n + \log_2 \log_2 \mathcal{C}_{\delta/2})$ bits.

Note that since $a + b = O(\max(a, b))$ for non-negative variables a and b , the upper bound from Theorem 12.13 is of the same order as the lower bound from Theorem 12.12.

It is known that both the packing and covering number of the unit sphere are of the order $\Theta(1/\delta^d)$. Substituting this into Theorems 12.12 and 12.13, we get the following corollaries.

Corollary 12.14. *Let S^{d-1} be the unit sphere in $\mathbb{R}^d$. Let $\mathcal{A}$ be a learning algorithm that can memorize any set of n points on S^{d-1} as long as every pair of points $\mathbf{x}_i, \mathbf{x}_j \in \mathcal{D}$ satisfies $\|\mathbf{x}_i - \mathbf{x}_j\| \geq \delta$. Then, we need $\Omega(\max(n, \log d + \log \log \frac{1}{\delta}))$ bits to represent the models output by $\mathcal{A}$.*

Corollary 12.15. *Let S^{d-1} be the unit sphere in $\mathbb{R}^d$. Then, there exists a learning algorithm $\mathcal{A}$ that can memorize any set of n points on S^{d-1} as long as every pair of feature vectors $\mathbf{x}_i, \mathbf{x}_j \in \mathcal{D}$ satisfies $\|\mathbf{x}_i - \mathbf{x}_j\| \geq \delta$, and the models returned by $\mathcal{A}$ can be represented in $O(n + \log d + \log \log \frac{1}{\delta})$ bits.*

Note that if we have a set of n points on the unit sphere separated by at least distance δ , then the set satisfies *both* Assumption 12.5 and Assumption 12.6. Hence, these two corollaries give bounds on the bit complexity of memorization under both the δ -separation assumptions.

12.7 Lower bound construction

For our lower bound construction, we will have n points on the unit sphere in d -dimensions. Each neuron in the first layer represents a hyperplane in d -dimensions. The main observation that we use is that for every pair of points $\mathbf{x}_i$ and $\mathbf{x}_j$ such that they have different labels, there should be a hyperplane passing in between them. Thus, we aim to lower bound the minimum number of hyperplanes needed for this to happen.

This is related to the problem of having to separate every point using hyperplanes, that is, the problem where we need to ensure that there is a hyperplane passing in between *every* pair of points $\mathbf{x}_i$ and $\mathbf{x}_j$. For this problem we have the following theorem.

Theorem 12.16. *For every $0 < \delta \leq \frac{1}{2}$ and some universal constants C_1 and C_2 , there exists a set of n points on the d -dimensional sphere, with $n \in \left[\frac{C_1 d^2 \log^5(d/\delta)}{\delta}, \left(\frac{C_2}{\delta} \right)^{\frac{d}{2}} \right]$, such that each pair of points is separated by a distance of at least δ , and one needs $\Omega\left(\frac{\log n}{\sqrt{\delta} \log \frac{\log n}{\delta}}\right)$ hyperplanes to separate them.*

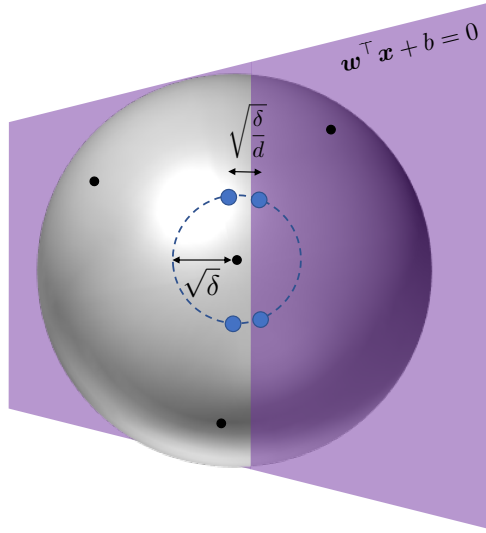


Figure 12.4: In this figure, the black dots represent the cluster centers and the blue circles represent the samples in a cluster. In high dimensions, the points in each cluster get concentrated within distance $\mathcal{O}(\sqrt{\delta/d})$ in any direction. Hence any hyperplane, that wants to separate a constant fraction of the points in a cluster, needs to pass within distance $\mathcal{O}(\sqrt{\delta/d})$ of the cluster center, as shown in the figure.

We use essentially the same construction for both the problems. On the sphere, first we sample $\sqrt{n}$ points uniformly at random, which we call *cluster centers*. Around each of these centers, we sample $\sqrt{n}$ points uniformly at random from the cap of radius $\sqrt{\delta}$, and call these sets of points *clusters*. Hence, in total we sample n points on the sphere and it can be shown that with high probability these will all be at least δ distance away from each other as long as $n \in \left[\frac{d^2}{\delta}, \frac{1}{\delta^{d/2}}\right]$. In high dimensions, it can be shown that for each cluster and in any direction, most points lie within a distance $\sqrt{\delta/d}$ of the cluster center. Hence, if a hyperplane aims to ‘effectively’ separate pairs of points in a cluster, it needs to pass within distance $\sqrt{\delta/d}$ of the cluster center. It can also be shown that in high dimensions any hyperplane can only pass within distance $\sqrt{\delta/d}$ of the cluster centers of $\sqrt{\delta}$ fraction of the clusters. Finally, we use the following necessary inequality to get a bound on the required number of hyperplanes:

(# hyperplanes needed)

$$\geq \frac{(\# \text{ clusters}) \times (\# \text{ hyperplanes needed per cluster})}{(\# \text{ clusters a hyperplane can 'effectively' separate})}.$$

We know that the number of clusters is $\sqrt{n}$, and we showed above that $(\# \text{ clusters a hyperplane can 'effectively' separate})$ is $\sqrt{\delta}$ fraction of the clusters, that is $\mathcal{O}(\sqrt{n\delta})$. Finally, for the term $(\# \text{ hyperplanes needed per cluster})$, we use 1 for the memorization problem, where we need to separate points with opposite label. For the problem of separating every pair of points, we know that $(\# \text{ hyperplanes needed per cluster}) = \Omega(\log_2 \sqrt{n})$ since there $\sqrt{n}$ points in each cluster and hence we need at least $\log_2 \sqrt{n}$ hyperplanes per cluster. Substituting these values gives us the bounds in Theorems 12.9 and 12.16.

Note that we skipped many details in the sketch above. The complete proof is provided in the appendix, and the construction there is slightly different.

12.8 Conclusion

In this work, we study the memorization capacity of threshold networks under the δ -separation assumption. We improve the existing bounds on the number of neurons and weights required, from exponential in δ , to almost linear. We also prove new, δ -dependent lower bounds for the memorization capacity of threshold networks, that together with our upper bound, shows that δ (the separation) indeed impacts the network architecture required for memorization.

Part IV

Appendices

A.1 Proof of Theorem 4.8

Theorem A.1 (Formal version). *Under Assumptions 4.1-4.7, let the step size of RR be*

$$\alpha = \frac{4l \log T}{T\mu}, \text{ where } l \leq 2$$

and the number of epochs be

$$K \geq 128 \frac{L^2}{\mu^2} \log T.$$

Then after T iterations RR,

$$\mathbb{E} \left[\|x_n^K - x^*\|^2 \right] \leq \frac{\|x_0^0 - x^*\|^2}{T^l} + \frac{2^{13} G^2 L^2 \log^3 T}{T^2 \mu^4} + \frac{2^{15} G^2 L^2 n^2 \log^4 T}{T^3 \mu^4}.$$

Proof. The proof for upper bound uses the framework of Haochen and Sra (2019a), combined with some crucial ideas from Nagaraj et al. (2019).

In the block diagram below we connect the pieces needed to establish the proof. All lemmas and proofs follow.

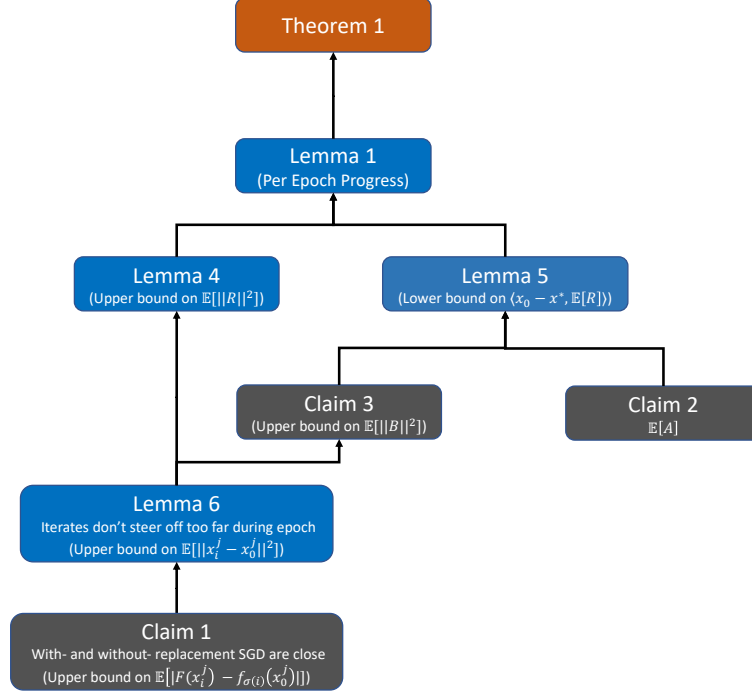


Figure A.1: A dependency graph for the proof of Theorem 4.8, giving short descriptions of the components required.

The proof strategy is to quantify the progress made during each epoch and then simply unrolling that for K epochs. Towards that end, we have the following lemma

Lemma A.2. *Let the RR step size be $\alpha = \frac{4l \log T}{T\mu}$ and the total number of epochs be $K \geq 128 \frac{L^2}{\mu^2} \log T$, where $l \leq 2$. Then for any epoch,*

$$\mathbb{E} [\|x_0^j - x^*\|^2] \leq \left(1 - \frac{n\alpha\mu}{4}\right) \|x_0^{j-1} - x^*\|^2 + 16n\alpha^3 G^2 L^2 \mu^{-1} + 20n^3 \alpha^4 G^2 L^2. \quad (4.1)$$

As mentioned before, we apply this lemma recursively to all epochs. For ease of notation, define $C_1 := 16G^2 L^2 \mu^{-1}$ and $C_2 := 20G^2 L^2$. Then,

$$\begin{aligned} \mathbb{E} [\|x_n^K - x^*\|^2] &\leq \left(1 - \frac{n\alpha\mu}{4}\right) \mathbb{E} [\|x_0^K - x^*\|^2] + C_1 n \alpha^3 + C_2 n^3 \alpha^4 \\ &\leq \left(1 - \frac{n\alpha\mu}{4}\right)^2 \mathbb{E} [\|x_0^{K-1} - x^*\|^2] + (C_1 n \alpha^3 + C_2 n^3 \alpha^4) \left(1 + \left(1 - \frac{n\alpha\mu}{4}\right)\right) \\ &\vdots \\ &\leq \left(1 - \frac{n\alpha\mu}{4}\right)^{K+1} \mathbb{E} [\|x_0^0 - x^*\|^2] + (C_1 n \alpha^3 + C_2 n^3 \alpha^4) \sum_{j=1}^K \left(1 - \frac{n\alpha\mu}{4}\right)^{j-1} \end{aligned}$$

$$= \left(1 - \frac{n\alpha\mu}{4}\right)^{K+1} \|x_0^0 - x^*\|^2 + (C_1 n\alpha^3 + C_2 n^3 \alpha^4) \sum_{j=1}^K \left(1 - \frac{n\alpha\mu}{4}\right)^{j-1}.$$

We can now use the fact that $(1 - x) \leq e^{-x}$ and $\left(1 - \frac{n\alpha\mu}{4}\right) \leq 1$, to get the following bound:

$$\mathbb{E} \left[\|x_n^K - x^*\|^2 \right] \leq e^{-\frac{n\alpha\mu}{4}K} \|x_0^0 - x^*\|^2 + (C_1 n\alpha^3 + C_2 n^3 \alpha^4)K.$$

By setting the stepsize to be $\alpha = \frac{8 \log T}{T\mu}$ and noting that $T = nK$, we get that

$$\begin{aligned} \mathbb{E} \left[\|x_n^K - x^*\|^2 \right] &\leq e^{-n \frac{4l \log T}{T\mu} \frac{\mu}{4} K} \|x_0^0 - x^*\|^2 + (n\alpha^3 C_1 + \alpha^4 n^3 C_2)K \\ &= e^{-l \log T} \|x_0^0 - x^*\|^2 + \frac{2^9 C_1 \log^3 T}{T^2 \mu^3} + \frac{2^{12} C_2 n^2 \log^4 T}{T^3 \mu^4}. \end{aligned}$$

Substituting the values of C_1 and C_2 back into the inequality gives

$$\mathbb{E} \left[\|x_n^K - x^*\|^2 \right] \leq \frac{\|x_0^0 - x^*\|^2}{T^l} + \frac{2^{13} G^2 L^2 \log^3 T}{T^2 \mu^4} + \frac{2^{15} G^2 L^2 n^2 \log^4 T}{T^3 \mu^4}.$$

□

A.1.1 Proof of Lemma 4.9

Proof. Throughout this proof we will be working inside an epoch, so we skip using the super script j in x_i^j which denotes the j -th epoch. Thus in this proof, x_0 refers to the iterate at beginning of that epoch. Let σ denote the permutation of $[n]$ used in this epoch. Therefore at the i -th iteration of this epoch, we take a descent step using the gradient of $f_{\sigma(i)}$.

Next we define the error term (same as the error term defined in (Haochen and Sra, 2019a))

$$R := \sum_{i=1}^n \left(\nabla f_{\sigma(i)}(x_{i-1}) - \nabla F(x_0) \right) = \sum_{i=1}^n \left(\nabla f_{\sigma(i)}(x_{i-1}) - \nabla f_{\sigma(i)}(x_0) \right). \quad (\text{A.1})$$

Then, using the iterative relation for gradient descent, we get

$$\begin{aligned} \|x_n - x^*\|^2 &= \left\| \left(x_0 - \sum_{i=1}^n \alpha \nabla f_{\sigma(i)}(x_{i-1}) \right) - x^* \right\|^2 \\ &= \|x_0 - x^*\|^2 - 2\alpha \left\langle x_0 - x^*, \sum_{i=1}^n \nabla f_{\sigma(i)}(x_{i-1}) \right\rangle + \alpha^2 \left\| \sum_{i=1}^n \nabla f_{\sigma(i)}(x_{i-1}) \right\|^2 \end{aligned}$$

$$\begin{aligned}
&= \|x_0 - x^*\|^2 - 2n\alpha \langle x_0 - x^*, \nabla F(x_0) \rangle - 2\alpha \langle x_0 - x^*, R \rangle \\
&\quad + \alpha^2 \|n\nabla F(x_0) + R\|^2 \\
&\stackrel{(a)}{\leq} \|x_0 - x^*\|^2 - 2n\alpha \langle x_0 - x^*, \nabla F(x_0) \rangle + 2\alpha^2 n^2 \|\nabla F(x_0)\|^2 \\
&\quad - 2\alpha \langle x_0 - x^*, R \rangle + 2\alpha^2 \|R\|^2 \\
&\stackrel{(b)}{\leq} \|x_0 - x^*\|^2 - 2n\alpha \left(\frac{\mu}{2} \|x_0 - x^*\|^2 + \frac{1}{2L} \|\nabla F(x_0)\|^2 \right) + 2\alpha^2 n^2 \|\nabla F(x_0)\|^2 \\
&\quad + 2\alpha^2 \|R\|^2 - 2\alpha \langle x_0 - x^*, R \rangle \\
&= (1 - n\alpha\mu) \|x_0 - x^*\|^2 - n\alpha \left(\frac{1}{L} - 2n\alpha \right) \|\nabla F(x_0)\|^2 + 2\alpha^2 \|R\|^2 \\
&\quad - 2\alpha \langle x_0 - x^*, R \rangle. \tag{A.2}
\end{aligned}$$

The inequality (a) above comes from the fact that $\|a + b\|^2 \leq 2\|a\|^2 + 2\|b\|^2$. For inequality (b), we used the property of strong convexity given by Theorem A.3 below:

Theorem A.3. [Nesterov (2013, Theorem 2.1.11)] For an L -smooth and μ -strongly convex function F ,

$$\langle \nabla F(x) - \nabla F(y), x - y \rangle \geq \frac{\mu}{2} \|x - y\|^2 + \frac{1}{2L} \|\nabla F(x) - \nabla F(y)\|^2$$

NOTE: We have slightly modified the original theorem by using the fact that $L \geq \mu$.

Thus, to upper bound the expected value of $\|x_n - x^*\|^2$, Ineq. (A.2) says that we need to control the expected magnitude of the error term, which is $\mathbb{E}[2\alpha^2 \|R\|^2]$; and its expected alignment with $(x_0 - x^*)$, which is $\mathbb{E}[-2\alpha \langle x_0 - x^*, R \rangle]$. To achieve that we introduce the following two lemmas.

Lemma A.4. If $\alpha \leq 1/2nL$, then the magnitude of the error is bounded above, in expectation:

$$\mathbb{E}[\|R\|^2] \leq L^2 n^2 \|x_0 - x^*\|^2 + 5n^3 \alpha^2 L^2 G^2.$$

Lemma A.5. If $\alpha \leq 1/2nL$, then the error term's alignment with $x_0 - x^*$ is bounded below, in expectation:

$$\begin{aligned}
\langle x_0 - x^*, \mathbb{E}[R] \rangle &\geq -\frac{\alpha n^2 \|\nabla F(x_0)\|^2}{2} - \left(\frac{n\mu}{4} + \frac{2\alpha^2 n^3 L^4}{\mu} \right) \|x_0 - x^*\|^2 \\
&\quad - \frac{10\alpha^4 L^4 G^2 n^4}{\mu} - \frac{8n\alpha^2 G^2 L^2}{\mu}.
\end{aligned}$$

For Lemma A.4 and Lemma A.5, we will show later (see Ineq. (A.5)) that our choice of parameters ensure $\alpha \leq 1/2nL$.

Substituting the inequalities from these two lemmas into Ineq. (A.2), we get that

$$\begin{aligned}
& \mathbb{E}[\|x_n - x^*\|^2] \\
& \leq (1 - n\alpha\mu) \|x_0 - x^*\|^2 - n\alpha \left(\frac{1}{2L} - 2n\alpha \right) \|\nabla F(x_0)\|^2 \\
& \quad + 2\alpha^2(L^2n^2\|x_0 - x^*\|^2 + 5n^3\alpha^2L^2G^2) \\
& \quad - 2\alpha \left(-\frac{\alpha n^2\|\nabla F(x_0)\|^2}{2} - \left(\frac{n\mu}{4} + \frac{2\alpha^2n^3L^4}{\mu} \right) \|x_0 - x^*\|^2 \right. \\
& \quad \quad \left. - \frac{10\alpha^4L^4G^2n^4}{\mu} - \frac{8n\alpha^2G^2L^2}{\mu} \right) \\
& = \left(1 - n\alpha\mu + 2\alpha^2L^2n^2 + \frac{\alpha\mu n}{2} + \frac{4n^3L^4\alpha^3}{\mu} \right) \|x_0 - x^*\|^2 \\
& \quad - n\alpha \left(\frac{1}{L} - 3n\alpha \right) \|\nabla F(x_0)\|^2 + 10n^3\alpha^4L^2G^2 + \frac{20\alpha^5L^4G^2n^4}{\mu} \\
& \quad + \frac{16n\alpha^3G^2L^2}{\mu} \\
& = \left(1 - \frac{n\alpha\mu}{2} + 2\alpha^2L^2n^2 + 4\mu^{-1}n^3L^4\alpha^3 \right) \|x_0 - x^*\|^2 \\
& \quad - n\alpha \left(\frac{1}{L} - 3n\alpha \right) \|\nabla F(x_0)\|^2 + 10n^3\alpha^4L^2G^2 + 20\mu^{-1}\alpha^5L^4G^2n^4 \\
& \quad + 16n\alpha^3G^2L^2\mu^{-1}. \tag{A.3}
\end{aligned}$$

We will prove the following inequalities shortly

$$\begin{aligned}
\frac{n\alpha\mu}{4} - 2\alpha^2L^2n^2 - 4\mu^{-1}n^3L^4\alpha^3 & \stackrel{(c)}{\geq} 0 \\
\frac{1}{L} - 3n\alpha & \stackrel{(d)}{\geq} 0 \\
10n^3\alpha^4L^2G^2 & \stackrel{(e)}{\geq} 20\mu^{-1}\alpha^5L^4G^2n^4.
\end{aligned}$$

Finally, using (c), (d) and (e) in Ineq. (A.3), we get

$$\mathbb{E}[\|x_n - x^*\|^2] \leq \left(1 - \frac{n\alpha\mu}{4} \right) \|x_0 - x^*\|^2 + 20n^3\alpha^4L^2G^2 + 16n\alpha^3G^2L^2\mu^{-1}.$$

This completes the proof. The only thing left is to prove the inequalities (c), (d) and (e), which we'll do next.

(c) and (d): It can be shown that $2\alpha^2L^2n^2 \geq 4\mu^{-1}n^3L^4\alpha^3$ (See (e) below). So to prove (c), it is sufficient to show that

$$\frac{n\alpha\mu}{4} \geq 4\alpha^2L^2n^2.$$

Recall that $\alpha = \frac{4l \log T}{T\mu} \leq \frac{8 \log T}{T\mu}$, $T = nK$, and $K \geq 128 \frac{L^2}{\mu^2} \log T$. Then,

$$\begin{aligned}
 \alpha &\leq \frac{8 \log T}{T\mu} \\
 &= \frac{8 \log T}{nK\mu} \\
 &\leq \frac{\mu^2 \log T}{16nL^2\mu \log T} \\
 &= \frac{\mu}{16nL^2}.
 \end{aligned} \tag{A.4}$$

This is equivalent to $\frac{n\alpha\mu}{4} \geq 4\alpha^2 L^2 n^2$. Thus, we have proven (c). To prove (d), we continue on the the series of inequalities:

$$\begin{aligned}
 \alpha &\leq \frac{\mu}{16nL^2} \\
 &\leq \frac{1}{16nL} \\
 &\leq \frac{1}{3nL}.
 \end{aligned} \tag{A.5}$$

This proves (d).

(e):

$$\begin{aligned}
 \alpha &\leq \frac{\mu}{16L^2n} && \text{[Using Ineq. (A.4)]} \\
 &\leq \frac{\mu}{2L^2n} \\
 \implies 10n^3\alpha^4 L^2 G^2 &\geq 20\mu^{-1}\alpha^5 L^4 G^2 n^4.
 \end{aligned}$$

□

A.1.2 Proof of Lemma A.4

Proof. Throughout this proof we will be working inside an epoch, so we skip using the super script j in x_i^j which denotes the j -th epoch. Thus in this proof, x_0 refers to the iterate at beginning of that epoch. Let σ denote the permutation of $[n]$ used in this epoch. Therefore at the i -th iteration of this epoch, we take a descent step using the gradient of $f_{\sigma(i)}$.

$$\mathbb{E} [\|R\|^2] = \mathbb{E} \left[\left\| \sum_{i=1}^n \left(\nabla f_{\sigma(i)}(x_{i-1}) - \nabla f_{\sigma(i)}(x_0) \right) \right\|^2 \right]$$

$$\leq \mathbb{E} \left[\left(\sum_{i=1}^n \left\| \nabla f_{\sigma(i)}(x_{i-1}) - \nabla f_{\sigma(i)}(x_0) \right\| \right)^2 \right]. \quad [\text{Triangle inequality}]$$

Now, if we can bound $\|x_{i-1} - x_0\|$, then we can also bound $\|\nabla f_{\sigma(i)}(x_{i-1}) - \nabla f_{\sigma(i)}(x_0)\|$ using gradient Lipschitzness. To bound $\|x_{i-1} - x_0\|$, we use Lemma A.6 below which says that if an epoch starts close to the minimizer x^* , then during that epoch, the iterates do not wander off too far away from the beginning of that epoch.

Lemma A.6. [Nagaraj et al. (2019, Lemma 5), but proved slightly differently] *If $\alpha \leq 2/L$ then,*

$$\begin{aligned} \mathbb{E}[\|x_i^j - x_0^j\|^2] &\leq \mathbb{E}[\|x_i^j - x_0^j\|^2] \leq 5i\alpha^2 G^2 + 2i\alpha(F(x_0^j) - F(x^*)) \\ &\leq 5i\alpha^2 G^2 + i\alpha L \|x_0^j - x^*\|^2. \end{aligned}$$

Thus, continuing the sequence of inequalities,

$$\begin{aligned} \mathbb{E} \left[\left(\sum_{i=1}^n \left\| \nabla f_{\sigma(i)}(x_{i-1}) - \nabla f_{\sigma(i)}(x_0) \right\| \right)^2 \right] &\leq L^2 \mathbb{E} \left[\left(\sum_{i=1}^n \|x_{i-1} - x_0\| \right)^2 \right] \\ &\quad [\text{Gradient Lipschitzness}] \\ &= L^2 \mathbb{E} \left[\sum_{i=1}^n \sum_{j=1}^n \|x_{i-1} - x_0\| \|x_{j-1} - x_0\| \right] \\ &= L^2 \sum_{i=1}^n \sum_{j=1}^n \mathbb{E} [\|x_{i-1} - x_0\| \|x_{j-1} - x_0\|] \\ &\leq L^2 \sum_{i=1}^n \sum_{j=1}^n \sqrt{\mathbb{E} [\|x_{i-1} - x_0\|^2]} \sqrt{\mathbb{E} [\|x_{j-1} - x_0\|^2]} \\ &\quad [\text{Cauchy-Schwartz inequality}] \\ &\leq L^2 n^2 (2n\alpha L \|x_0 - x^*\|^2 + 5n\alpha^2 G^2) \quad [\text{Using Lemma A.6}] \\ &\leq L^2 n^2 \|x_0 - x^*\|^2 + 5n^3 \alpha^2 G^2 L^2, \quad (\text{A.6}) \end{aligned}$$

where we used the assumption that $\alpha \leq 1/2nL$ in the last step. □

A.1.3 Proof of Lemma A.5

Proof. Throughout this proof we will be working inside an epoch, so we skip using the super script j in x_i^j which denotes the j -th epoch. Thus in this proof, x_0 refers to the iterate at beginning of that epoch. Let σ denote the permutation of $[n]$ used in this epoch.

Therefore at the i -th iteration of this epoch, we take a descent step using the gradient of $f_{\sigma(i)}$.

$$\begin{aligned}
R &= \sum_{i=1}^n \left[\nabla f_{\sigma(i)}(x_{i-1}) - \nabla f_{\sigma(i)}(x_0) \right] \\
&= \sum_{i=1}^n \left[\nabla f_{\sigma(i)} \left(x_0 - \alpha \sum_{j=1}^{i-1} \nabla f_{\sigma(j)}(x_{j-1}) \right) - \nabla f_{\sigma(i)}(x_0) \right] \\
&\quad \text{[By the definition of RR iterations: } x_{i+1} = x_i - \alpha \nabla f_{\sigma(i)}(x_i)\text{]} \\
&= \sum_{i=1}^n \left[\nabla f_{\sigma(i)} \left(x_0 - \alpha \sum_{j=1}^{i-1} \nabla f_{\sigma(j)}(x_{j-1}) \right) - \nabla f_{\sigma(i)}(x_0) \right. \\
&\quad \left. + \nabla f_{\sigma(i)} \left(x_0 - \alpha \sum_{j=1}^{i-1} \nabla f_{\sigma(j)}(x_0) \right) - \nabla f_{\sigma(i)} \left(x_0 - \alpha \sum_{j=1}^{i-1} \nabla f_{\sigma(j)}(x_0) \right) \right] \\
&= \sum_{i=1}^n \left[\nabla f_{\sigma(i)} \left(x_0 - \alpha \sum_{j=1}^{i-1} \nabla f_{\sigma(j)}(x_0) \right) - \nabla f_{\sigma(i)}(x_0) \right] \\
&\quad + \sum_{i=1}^n \left[\nabla f_{\sigma(i)} \left(x_0 - \alpha \sum_{j=1}^{i-1} \nabla f_{\sigma(j)}(x_{j-1}) \right) - \nabla f_{\sigma(i)} \left(x_0 - \alpha \sum_{j=1}^{i-1} \nabla f_{\sigma(j)}(x_0) \right) \right] \\
&= A + B, \tag{A.7}
\end{aligned}$$

where

$$A := \sum_{i=1}^n \left[\nabla f_{\sigma(i)} \left(x_0 - \alpha \sum_{j=1}^{i-1} \nabla f_{\sigma(j)}(x_0) \right) - \nabla f_{\sigma(i)}(x_0) \right],$$

and

$$B := \sum_{i=1}^n \left[\nabla f_{\sigma(i)} \left(x_0 - \alpha \sum_{j=1}^{i-1} \nabla f_{\sigma(j)}(x_{j-1}) \right) - \nabla f_{\sigma(i)} \left(x_0 - \alpha \sum_{j=1}^{i-1} \nabla f_{\sigma(j)}(x_0) \right) \right].$$

A and B are the same terms as the ones defined in Haochen and Sra (2019a). The difference in our analysis and Haochen and Sra (2019a) is that we get tighter bounds on these terms.

In the following, we use u to denote a random vector with norm less than or equal to 1. Also, assume that H is the Hessian of the quadratic function F .

Claim A.7.

$$\mathbb{E}[A] = (2n\alpha GL)u - \alpha \frac{n(n-1)}{2} H \nabla F(x_0).$$

Claim A.8. *If $\alpha \leq 1/2nL$, then*

$$\|\mathbb{E}[B]\|^2 \leq n^4 L^4 \alpha^2 \|x_0 - x^*\|^2 + 5n^5 L^4 \alpha^4 G^2.$$

Using Eq. (A.7) and Claim A.7:

$$\begin{aligned} \langle x_0 - x^*, \mathbb{E}[R] \rangle &= \langle x_0 - x^*, \mathbb{E}[A] + \mathbb{E}[B] \rangle \\ &= \langle x_0 - x^*, \mathbb{E}[A] \rangle + \langle x_0 - x^*, \mathbb{E}[B] \rangle \\ &= -\alpha \frac{n(n-1)}{2} \langle x_0 - x^*, H \nabla F(x_0) \rangle \\ &\quad + \langle (2n\alpha GL)u, x_0 - x^* \rangle + \langle x_0 - x^*, \mathbb{E}[B] \rangle \\ &= -\alpha \frac{n(n-1)}{2} \|\nabla F(x_0)\|^2 + \langle (2n\alpha GL)u, x_0 - x^* \rangle \\ &\quad + \langle x_0 - x^*, \mathbb{E}[B] \rangle. \end{aligned} \tag{A.8}$$

For the middle term in Eq. (A.8), we use Cauchy-Schwarz inequality and the AM-GM inequality as follows

$$\begin{aligned} \langle (2n\alpha GL)u, x_0 - x^* \rangle &\geq -\|(2n\alpha GL)u\| \|x_0 - x^*\| \\ &\geq -\left[\frac{\lambda}{2} \|x_0 - x^*\|^2 + \frac{1}{2\lambda} (2n\alpha GL)^2 \right], \end{aligned} \tag{A.9}$$

where λ is a positive number.

We bound the last term in Eq. (A.8) similarly,

$$\begin{aligned} \langle x_0 - x^*, \mathbb{E}[B] \rangle &\geq -\|x_0 - x^*\| \|\mathbb{E}[B]\| \\ &\geq -\left[\frac{\lambda}{2} \|x_0 - x^*\|^2 + \frac{1}{2\lambda} \|\mathbb{E}[B]\|^2 \right]. \end{aligned} \tag{A.10}$$

Setting $\lambda = \frac{1}{4}\mu n$ and continuing on from Eq. (A.8), we get

$$\begin{aligned} \langle x_0 - x^*, \mathbb{E}[R] \rangle &= -\alpha \frac{n(n-1)}{2} \|\nabla F(x_0)\|^2 + \langle (2n\alpha GL)u, x_0 - x^* \rangle + \langle x_0 - x^*, \mathbb{E}[B] \rangle \\ &\geq -\frac{1}{2} \alpha n^2 \|\nabla F(x_0)\|^2 - \left[\frac{\lambda}{2} \|x_0 - x^*\|^2 + \frac{1}{2\lambda} (2n\alpha GL)^2 \right] \\ &\quad - \left[\frac{\lambda}{2} \|x_0 - x^*\|^2 + \frac{1}{2\lambda} \|\mathbb{E}[B]\|^2 \right] \quad [\text{Using Ineq. (A.9) and (A.10)}] \\ &= -\frac{1}{2} \alpha n^2 \|\nabla F(x_0)\|^2 - \frac{1}{4} \mu n \|x_0 - x^*\|^2 - \frac{2}{\mu n} (2n\alpha GL)^2 - \frac{2}{\mu n} \|\mathbb{E}[B]\|^2 \end{aligned}$$

$$\begin{aligned}
&\geq -\frac{\alpha n^2 \|\nabla F(x_0)\|^2}{2} - \left(\frac{n\mu}{4} + 2\alpha^2 n^3 \mu^{-1} L^4\right) \|x_0 - x^*\|^2 - 10\mu^{-1} \alpha^4 L^4 G^2 n^4 \\
&\quad - 8n\alpha^2 G^2 L^2 \mu^{-1}.
\end{aligned}$$

[Using Claim A.8]

□

A.1.3.1 Proof of Claim A.7

Proof. The proof strategy of this claim is to use the fact that the Hessian of a quadratic function is constant, and thus we can use it calculate exactly the gradient difference of a quadratic function. In this proof, we will use u to denote a random variable with norm at most 1.

$$\begin{aligned}
\mathbb{E}[A] &= \mathbb{E} \left[\sum_{i=1}^n \left[\nabla f_{\sigma(i)} \left(x_0 - \alpha \sum_{j=1}^{i-1} \nabla f_{\sigma(j)}(x_0) \right) - \nabla f_{\sigma(i)}(x_0) \right] \right] \\
&= \sum_{i=1}^n \mathbb{E} \left[\nabla f_{\sigma(i)} \left(x_0 - \alpha \sum_{j=1}^{i-1} \nabla f_{\sigma(j)}(x_0) \right) - \nabla f_{\sigma(i)}(x_0) \right] \\
&\stackrel{(f)}{=} \sum_{i=1}^n \mathbb{E} \left[\nabla F \left(x_0 - \alpha \sum_{j=1}^{i-1} \nabla f_{\sigma(j)}(x_0) \right) + (2\alpha GL)u - \nabla F(x_0) \right] \\
&= (2n\alpha GL)u + \sum_{i=1}^n \mathbb{E} \left[\nabla F \left(x_0 - \alpha \sum_{j=1}^{i-1} \nabla f_{\sigma(j)}(x_0) \right) - \nabla F(x_0) \right] \\
&= (2n\alpha GL)u - \alpha \sum_{i=1}^n \mathbb{E} \left[H \left(\sum_{j=1}^{i-1} \nabla f_{\sigma(j)}(x_0) \right) \right] \\
&\quad \text{[For quadratics, } \nabla F(x) - \nabla F(y) = H(x - y)\text{]} \\
&= (2n\alpha GL)u - \alpha H \sum_{i=1}^n \left(\sum_{j=1}^{i-1} \mathbb{E} [\nabla f_{\sigma(j)}(x_0)] \right) \\
&= (2n\alpha GL)u - \alpha \frac{n(n-1)}{2} H \nabla F(x_0).
\end{aligned}$$

We used Claim A.9 for (f). This concludes the proof of Claim A.7, and the proof of Claim A.9 is provided next.

Claim A.9.

$$\mathbb{E} \left[\nabla f_{\sigma(i)} \left(x_0 - \alpha \sum_{j=1}^{i-1} \nabla f_{\sigma(j)}(x_0) \right) \right] = \mathbb{E} \left[\nabla F \left(x_0 - \alpha \sum_{j=1}^{i-1} \nabla f_{\sigma(j)}(x_0) \right) \right] + (2\alpha GL)u$$

□

Proof of Claim A.9 The proof for this claim uses the iterate coupling technique.

Proof. Since $\forall i, j : \mathbb{P}(\sigma(i) = j) = 1/n$,

$$\begin{aligned} \mathbb{E} \left[\nabla f_{\sigma(i)} \left(x_0 - \alpha \sum_{j=1}^{i-1} \nabla f_{\sigma(j)}(x_0) \right) \right] \\ = \frac{1}{n} \sum_{s=1}^n \mathbb{E} \left[\nabla f_{\sigma(i)} \left(x_0 - \alpha \sum_{j=1}^{i-1} \nabla f_{\sigma(j)}(x_0) \right) \middle| \sigma(i) = s \right] \\ = \frac{1}{n} \sum_{s=1}^n \mathbb{E} \left[\nabla f_s \left(x_0 - \alpha \sum_{j=1}^{i-1} \nabla f_{\sigma(j)}(x_0) \right) \middle| \sigma(i) = s \right]. \end{aligned}$$

Note that the distribution of $(\sigma | \sigma(i) = s)$ can be created from the distribution of $(\sigma | \sigma(i) = 1)$ by taking all permutations from $\sigma | \sigma(i) = 1$ and swapping 1 and s in the permutations (this is essentially a coupling between the two distributions, same as the one in (Nagaraj et al., 2019)). This means that when we convert a permutation from $(\sigma | \sigma(i) = 1)$ to $(\sigma | \sigma(i) = s)$ in this manner, the sum $\sum_{j=1}^{i-1} \nabla f_{\sigma(j)}(x_0)$ would have a change of at most one component $f_{\sigma(j)}$ before and after the swap, and furthermore the component will have a norm of at most G . Thus because of the swap (adding a component and removing one), the norm changes by at most $2G$. In the following, we use $u_p, v_{(p,q)}, w_{(p,q)}$ and u to denote random vectors with norms at most 1. Hence, the sum $(\sum_{j=1}^{i-1} \nabla f_{\sigma(j)}(x_0) | \sigma(i) = s)$ is equal to $(2Gv_{(1,s)} + \sum_{j=1}^{i-1} \nabla f_{\sigma(j)}(x_0) | \sigma(i) = 1)$. Then, continuing on the sequence of equalities,

$$\begin{aligned} \mathbb{E} \left[\nabla f_{\sigma(i)} \left(x_0 - \alpha \sum_{j=1}^{i-1} \nabla f_{\sigma(j)}(x_0) \right) \right] \\ = \frac{1}{n} \sum_{s=1}^n \mathbb{E} \left[\nabla f_s \left(x_0 - \alpha \sum_{j=1}^{i-1} \nabla f_{\sigma(j)}(x_0) \right) | \sigma(i) = s \right] \\ = \frac{1}{n} \sum_{s=1}^n \mathbb{E} \left[\nabla f_s \left(x_0 - \alpha \sum_{j=1}^{i-1} \nabla f_{\sigma(j)}(x_0) + 2\alpha G v_{(1,s)} \right) | \sigma(i) = 1 \right] \\ = \frac{1}{n} \sum_{s=1}^n \mathbb{E} \left[\nabla f_s \left(x_0 - \alpha \sum_{j=1}^{i-1} \nabla f_{\sigma(j)}(x_0) \right) + (2\alpha GL) w_{(1,s)} | \sigma(i) = 1 \right] \\ \hspace{15em} [\text{Using gradient Lipschitzness}] \\ = \mathbb{E} \left[\nabla F \left(x_0 - \alpha \sum_{j=1}^{i-1} \nabla f_{\sigma(j)}(x_0) \right) | \sigma(i) = 1 \right] + (2\alpha GL) u_1. \end{aligned}$$

Similarly, for any s :

$$\begin{aligned} & \mathbb{E} \left[\nabla f_{\sigma(i)} \left(x_0 - \alpha \sum_{j=1}^{i-1} \nabla f_{\sigma(j)} (x_0) \right) \right] \\ &= \mathbb{E} \left[\nabla F \left(x_0 - \alpha \sum_{j=1}^{i-1} \nabla f_{\sigma(j)} (x_0) \right) \mid \sigma(i) = s \right] + (2\alpha GL)u_s. \end{aligned}$$

Hence,

$$\begin{aligned} & \mathbb{E} \left[\nabla f_{\sigma(i)} \left(x_0 - \alpha \sum_{j=1}^{i-1} \nabla f_{\sigma(j)} (x_0) \right) \right] \\ &= \frac{1}{n} \sum_{s=1}^n \left(\mathbb{E} \left[\nabla F \left(x_0 - \alpha \sum_{j=1}^{i-1} \nabla f_{\sigma(j)} (x_0) \right) \mid \sigma(i) = s \right] + (2\alpha GL)u_s \right) \\ &= \mathbb{E} \left[\nabla F \left(x_0 - \alpha \sum_{j=1}^{i-1} \nabla f_{\sigma(j)} (x_0) \right) \right] + (2\alpha GL)u. \end{aligned}$$

□

A.1.3.2 Proof of Claim A.8

Proof.

$$\begin{aligned} \|\mathbb{E}[B]\|^2 &\leq (\mathbb{E} [\|B\|])^2 && \text{[Jensen's inequality]} \\ &= \left(\mathbb{E} \left[\left\| \sum_{i=1}^n \left[\nabla f_{\sigma(i)} \left(x_0 - \alpha \sum_{j=1}^{i-1} \nabla f_{\sigma(j)} (x_{j-1}) \right) - \nabla f_{\sigma(i)} \left(x_0 - \alpha \sum_{j=1}^{i-1} \nabla f_{\sigma(j)} (x_0) \right) \right] \right\| \right] \right)^2 \\ &\leq \left(\mathbb{E} \left[\sum_{i=1}^n \left\| \nabla f_{\sigma(i)} \left(x_0 - \alpha \sum_{j=1}^{i-1} \nabla f_{\sigma(j)} (x_{j-1}) \right) - \nabla f_{\sigma(i)} \left(x_0 - \alpha \sum_{j=1}^{i-1} \nabla f_{\sigma(j)} (x_0) \right) \right\| \right] \right)^2 && \text{[Triangle inequality]} \\ &\leq \left(\mathbb{E} \left[\sum_{i=1}^n L\alpha \left\| \sum_{j=1}^{i-1} \nabla f_{\sigma(j)} (x_{j-1}) - \nabla f_{\sigma(j)} (x_0) \right\| \right] \right)^2 && \text{[Gradient Lipschitzness]} \\ &\leq L^2\alpha^2 \left(\mathbb{E} \left[\sum_{i=1}^n \sum_{j=1}^{i-1} \left\| \nabla f_{\sigma(j)} (x_{j-1}) - \nabla f_{\sigma(j)} (x_0) \right\| \right] \right)^2 && \text{[Triangle inequality]} \end{aligned}$$

$$\begin{aligned}
&\leq L^2 \alpha^2 \left(\mathbb{E} \left[\sum_{i=1}^n \sum_{j=1}^n \|\nabla f_{\sigma(j)}(x_{j-1}) - \nabla f_{\sigma(j)}(x_0)\| \right] \right)^2 \\
&= n^2 L^2 \alpha^2 \left(\mathbb{E} \left[\sum_{j=1}^n \|\nabla f_{\sigma(j)}(x_{j-1}) - \nabla f_{\sigma(j)}(x_0)\| \right] \right)^2 \\
&\leq n^2 L^2 \alpha^2 \mathbb{E} \left[\left(\sum_{j=1}^n \|\nabla f_{\sigma(j)}(x_{j-1}) - \nabla f_{\sigma(j)}(x_0)\| \right)^2 \right]. \quad \text{[Jensen's inequality]}
\end{aligned}$$

We proved an upper bound on $\mathbb{E} \left[\left(\sum_{j=1}^n \|\nabla f_{\sigma(j)}(x_{j-1}) - \nabla f_{\sigma(j)}(x_0)\| \right)^2 \right]$ in the proof of Lemma A.4 in Subsection A.1.2 under exactly the same assumptions as the ones for this claim (See Ineq. (A.6) in that proof). In particular, there we have proved that

$$\mathbb{E} \left[\left(\sum_{j=1}^n \|\nabla f_{\sigma(j)}(x_{j-1}) - \nabla f_{\sigma(j)}(x_0)\| \right)^2 \right] \leq L^2 n^2 \|x_0 - x^*\|^2 + 5n^3 \alpha^2 G^2 L^2.$$

Substituting this inequality in the set of inequalities above gives us the result. $\square$

A.1.4 Proof of Lemma A.6

We will only prove $\mathbb{E}[\|x_i^j - x_0^j\|^2] \leq 5i\alpha^2 + 2i\alpha(F(x_0^j) - F(x^*))$, the rest follows from Jensen's inequality and gradient Lipschitzness.

We will use the following claim, which is just Nagaraj et al. (2019, Lemma 4) proved in a slightly different way: we skip the Wasserstein framework but use the same coupling.

Claim A.10. [Nagaraj et al. (2019, Lemma 4)] *If $\alpha \leq \frac{2}{L}$, then for any epoch j and i -th ordered iterate during that epoch*

$$\left| \mathbb{E} \left[F(x_i^j) - f_{\sigma(j)}(x_i^j) \mid x_0^j \right] \right| \leq 2\alpha G^2. \quad (4.2)$$

The rest of the proof is identical to the proof in (Nagaraj et al., 2019). Because in this proof we work inside an epoch, so we skip the super script in the notation.

$$\begin{aligned}
\|x_{i+1} - x_0\|^2 &= \|x_i - x_0\|^2 - 2\alpha \langle \nabla f_{\sigma(i)}(x_i), x_i - x_0 \rangle + \alpha^2 \|\nabla f_{\sigma(i)}(x_i)\|^2 \\
&\leq \|x_i - x_0\|^2 - 2\alpha \langle \nabla f_{\sigma(i)}(x_i), x_i - x_0 \rangle + \alpha^2 G^2 \quad \text{[Bounded gradients]} \\
&\leq \|x_i - x_0\|^2 + 2\alpha (f_{\sigma(i)}(x_0) - f_{\sigma(i)}(x_j)) + \alpha^2 G^2,
\end{aligned}$$

where the last inequality is due to the convexity of $f_{\sigma(i)}$. Taking expectation both sides:

$$\begin{aligned}
\mathbb{E}[\|x_{i+1} - x_0\|^2 | x_0] &\leq \mathbb{E}[\|x_i - x_0\|^2 | x_0] + 2\alpha \mathbb{E}[f_{\sigma(i)}(x_0) - f_{\sigma(i)}(x_j) | x_0] + \alpha^2 G^2 \\
&= \mathbb{E}[\|x_i - x_0\|^2 | x_0] + 2\alpha F(x_0) + 2\alpha \mathbb{E}[-f_{\sigma(i)}(x_j) | x_0] + \alpha^2 G^2 \\
&= \mathbb{E}[\|x_i - x_0\|^2 | x_0] + 2\alpha F(x_0) \\
&\quad + 2\alpha \mathbb{E}[F(x_j) - f_{\sigma(i)}(x_j) - F(x_j) | x_0] + \alpha^2 G^2 \\
&\leq \mathbb{E}[\|x_i - x_0\|^2 | x_0] + 2\alpha F(x_0) \\
&\quad + 2\alpha \mathbb{E}[F(x_j) - f_{\sigma(i)}(x_j) - F(x^*) | x_0] + \alpha^2 G^2 \\
&\hspace{15em} [\text{Since } x^* \text{ is the minimizer of } F] \\
&= \mathbb{E}[\|x_i - x_0\|^2 | x_0] + 2\alpha(F(x_0) - F(x^*)) \\
&\quad + 2\alpha \mathbb{E}[F(x_j) - f_{\sigma(i)}(x_j) | x_0] + \alpha^2 G^2 \\
&\leq \mathbb{E}[\|x_i - x_0\|^2 | x_0] + 2\alpha(F(x_0) - F(x^*)) + 2\alpha(2\alpha G^2) + \alpha^2 G^2 \\
&\hspace{15em} [\text{Using Claim 4.11}] \\
&= \mathbb{E}[\|x_i - x_0\|^2 | x_0] + 2\alpha(F(x_0) - F(x^*)) + 5\alpha^2 G^2.
\end{aligned}$$

Unrolling this for i iterations gives us the required result.

A.1.4.1 Proof of Claim 4.11

The proof for this claim also uses the iterate coupling technique, similar to the proof of Claim A.9.

Proof. As written in the claim statement, we assume that the start of the epoch, x_0^j is given. In this proof, we work inside an epoch, so we skip the superscript j in x_i^j .

$$\begin{aligned}
\mathbb{E}[f_{\sigma(i)}(x_i)] &= \frac{1}{n} \sum_{s=1}^n \mathbb{E}[f_{\sigma(i)}(x_i) | \sigma(i) = s] \\
&= \frac{1}{n} \sum_{s=1}^n \mathbb{E}[f_s(x_i) | \sigma(i) = s].
\end{aligned}$$

Note that the distribution of $(\sigma | \sigma(i) = s)$ can be created from the distribution of $(\sigma | \sigma(i) = 1)$ by taking all permutations from $(\sigma | \sigma(i) = 1)$ and swapping 1 and s in the permutations (this is essentially a coupling between the two distributions, same as the one in (Nagaraj et al., 2019)). This means that when we convert a permutation from the distribution $(\sigma | \sigma(i) = 1)$ to a permutation from the distribution $(\sigma | \sigma(i) = s)$ in this manner, the corresponding $(x_i | \sigma(i) = 1)$ and $(x_i | \sigma(i) = s)$ would be within a distance of $2\alpha G$. Here is why this is true: let x' be an iterate reached using a permutation σ' from the distribution $(\sigma | \sigma(i) = 1)$. Now,

create σ'' by swapping 1 and s in σ' . Then $\sigma''(i) = s$ and hence it lies in the distribution $(\sigma|\sigma(i) = s)$. Let x'' be an iterate reached using σ'' . Then can use Lemma 2 from (Nagaraj et al., 2019), adapted to our setting:

Lemma A.11. [Nagaraj et al. (2019, Lemma 2)] Let $\alpha \leq 2/L$. Then almost surely, $\forall i \in [n]$,

$$\|x' - x''\| \leq 2G\alpha.$$

In the following, we use $v_{(p,q)}$ to denote a random vector with norm less than or equal to 1; and $w_{(p,q)}$, u_p and u to denote a random scalar with absolute value less than or equal to 1.

Then using Lemma A.11, $(x_i|\sigma(i) = s)$ is equal to $(x_i + v_{(1,s)}|\sigma(i) = 1)$ (Similar to what we did in the proof of Claim A.9.).

$$\begin{aligned} \mathbb{E} [f_{\sigma(i)}(x_i)] &= \frac{1}{n} \sum_{s=1}^n \mathbb{E} [f_s(x_i) | \sigma(i) = s] \\ &= \frac{1}{n} \sum_{s=1}^n \mathbb{E} [f_s(x_i + (2\alpha G)v_{(1,s)}) | \sigma(i) = 1] \\ &= \frac{1}{n} \sum_{s=1}^n \mathbb{E} [f_s(x_i) + (2\alpha G^2)w_{(1,s)} | \sigma(i) = 1] \\ &= \mathbb{E} [F(x_i) | \sigma(i) = 1] + (2\alpha G^2)u_1. \end{aligned}$$

Similarly, for any s :

$$\mathbb{E} [f_{\sigma(i)}(x_i)] = \mathbb{E} [F(x_i) | \sigma(i) = s] + (2\alpha G^2)u_s.$$

Hence,

$$\begin{aligned} \mathbb{E} [f_{\sigma(i)}(x_i)] &= \frac{1}{n} \sum_{s=1}^n (\mathbb{E} [F(x_i) | \sigma(i) = s] + (2\alpha G^2)u_s) \\ &= \mathbb{E} [F(x_i)] + (2\alpha G^2)u. \end{aligned}$$

All the calculations in this proof assumed that the initial point of the epoch, x_0^j is known. Thus, the equation above implies

$$\left| \mathbb{E} [F(x_i^j) - f_{\sigma(i)}(x_i^j) | x_0^j] \right| \leq 2\alpha G^2.$$

□

A.2 Proof of Theorem 4.12

Theorem A.12 (Formal version). *There exists an initialization point x_0^1 and a 1-strongly convex function F that is the mean of n smooth convex functions which have L -Lipschitz gradients ($L \geq 2^{17}$), such that if*

$$\frac{1}{nK} \leq \alpha \leq \frac{2^{-14}}{nL}, \quad K \geq 2^{14}L,$$

$n \geq 256$ and n is a multiple of 4, then,

$$\mathbb{E}[\|x_T - x^*\|^2] \geq \frac{2^{-56}G^2n}{T^2}.$$

Remarks:

1. Because $\mu = 1$, the condition number is just L . Note that the lower bound provided above is independent of L .
2. The theorem and proof have not been optimized with respect to the dependence on universal constants and it can probably be much better. In particular, the experiments in Subsection 4.5.1 use the same construction of functions with much better values of constants.

Proof. As with Theorem 4.8, we first start with a block diagram of the components required to establish the lower bound.

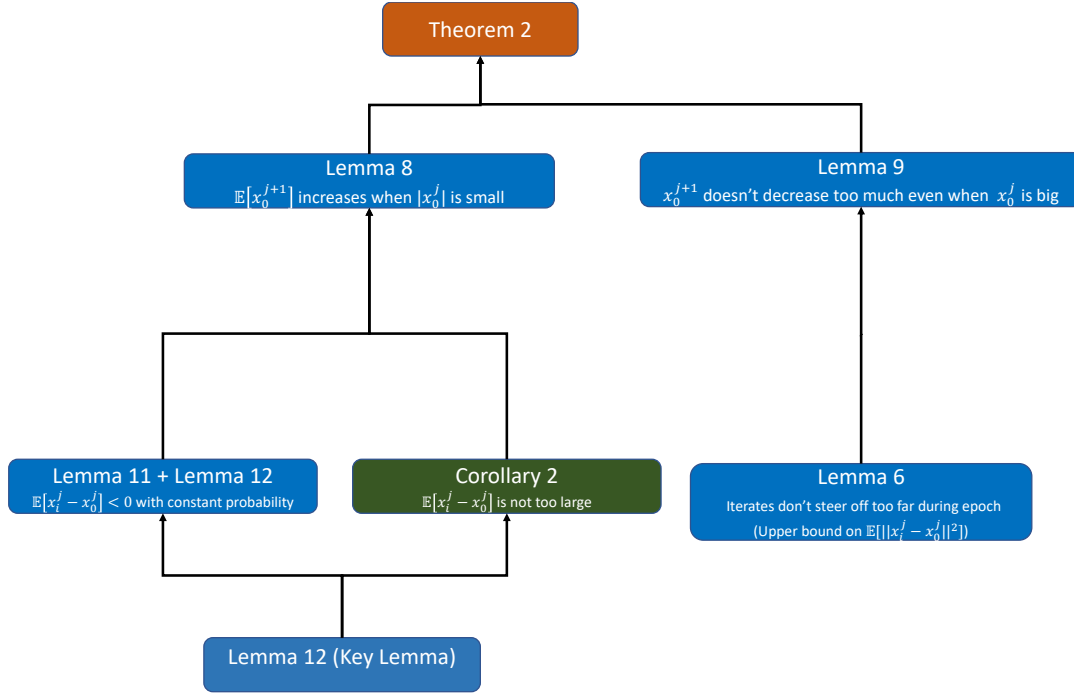


Figure A.2: A dependency graph for the proof of Theorem 4.12, giving short descriptions of the components for the proof of Theorem 4.12.

The function $F(x) = \frac{1}{n} \sum_{i=1}^n f_i(x)$ that we construct is of the form

$$F(x) = \begin{cases} \frac{x^2}{2} & \text{if } x \geq 0 \\ \frac{Lx^2}{2} & \text{if } x < 0. \end{cases}$$

where $n \geq 8$ and it is a multiple of 4. Of the n component functions f_i , half of them are defined as follows (we call these as functions of first kind):

$$\text{if } i \leq \frac{n}{2}, \text{ then } f_i(x) = \begin{cases} \frac{x^2}{2} + \frac{Gx}{2}, & \text{if } x \geq 0 \\ \frac{Lx^2}{2} + \frac{Gx}{2}, & \text{if } x < 0, \end{cases}$$

and the other half of the functions are defined as follows (we call these as functions of second kind):

$$\text{if } i > \frac{n}{2}, \text{ then } f_i(x) = \begin{cases} \frac{x^2}{2} - \frac{Gx}{2}, & \text{if } x \geq 0 \\ \frac{Lx^2}{2} - \frac{Gx}{2}, & \text{if } x < 0. \end{cases}$$

Let σ^j be the permutation of functions f_i 's that is used in the j -th epoch. Then, σ^j can

be represented by a permutation of the following multiset:

$$\underbrace{\{+1, \dots, +1\}}_{\frac{n}{2} \text{ times}}, \underbrace{\{-1, \dots, -1\}}_{\frac{n}{2} \text{ times}}$$

Accordingly, if $\sigma_i^j = +1$, we assume in the i -th iteration of the j -th epoch, a function of the 1st kind was sampled. Similarly if $\sigma_i^j = -1$, we assume in the i -th iteration of the j -th epoch, a function of the 2nd kind was sampled.

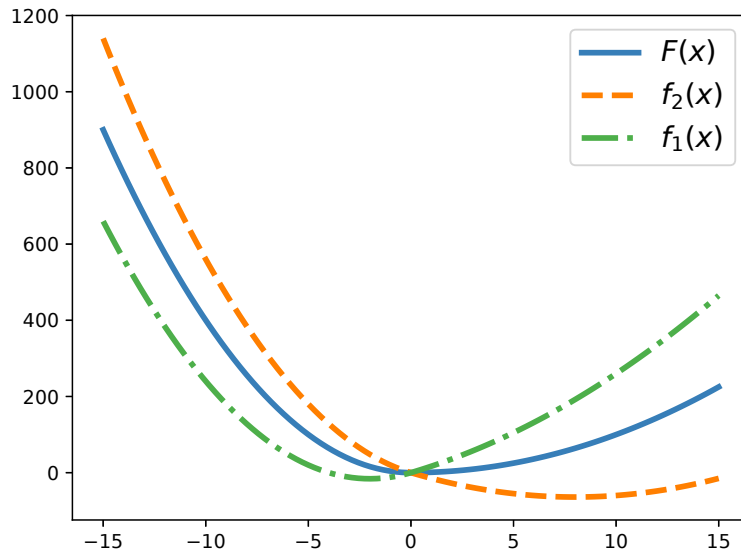


Figure A.3: Lower bound construction. Note that $f_1(x)$ represents functions of the first kind, and $f_2(x)$ represents functions of the second kind, and $F(x)$ represents the overall function.

Assumptions 4.1-4.6, except Assumptions 4.3 and 4.4 have been already proved for this function in the main text of this chapter. Thus, we only prove 4.3 and 4.4 here.

We will initialize at $x_0^1 = 0$. Next we prove that Assumption 4.4 is satisfied for our lowerbound construction. The minimizer of the functions of first kind is at $x = -G/2L$, and the minimizer of the functions of the second kind is at $x = G/2$. Between these two values, the norm of gradient of any of the two function kinds is always less than G . Thus, it is sufficient show that the iterates stay between the two minimizers. The step size we have chosen is small enough (smaller than $1/L$) to ensure that the iterates do not go outside the two minimizers. To see why this is so, consider the case when one is doing gradient descent on $g(x) = ax^2/2$. If the step length $\alpha < 1/a$, then the next iterate

$x_{t+1} = x_t - \alpha(ax_t) = x_t(1 - \alpha a)$ and the current iterate x_t lie on the same side of the minimizer (which is $x^* = 0$), that is the iterates never ‘cross over’ the minimizer. A similar logic here implies that iterates in our case stay within $[-G/2L, G/2]$. Thus gradient norm would never exceed G .

Next, we have the following key lemma of the proof:

Lemma A.13. *If $n \geq 256$ is a multiple of 4, $L \geq 2^{17}$, $\alpha \leq \frac{2^{-11}}{nL}$ and $|x_0^j| \leq 2^{-9}G\alpha\sqrt{n}$, then*

$$\mathbb{E} [x_0^{j+1}] \geq x_0^j + 2^{-12}LG\alpha^2n\sqrt{n}$$

Intuitively, what Lemma A.13 implies is that $\mathbb{E}[|x_0^{j+1}|]$ should keep increasing at rate $\Omega(\alpha^2n\sqrt{n})$ until $|x_0^j| = \Omega(\alpha\sqrt{n})$. This is exactly what we need, because for the range of step length α specified in the theorem statement, $|x_0^j| = \Omega(\alpha\sqrt{n})$ is the claimed error lower bound. The rest of the proof is just making this intuition rigorous.

The following helper lemma says that $\mathbb{E}[|x_0^{j+1}|]$ does not decrease by too much, even if $|x_0^j| > 2^{-9}G\alpha\sqrt{n}$.

Lemma A.14. *If $\alpha \leq 1/nL$, then*

$$\mathbb{E} [|x_0^{j+1}|] \geq |x_0^j|(1 - 2L\alpha n) - LG\alpha^2n\sqrt{5n},$$

and further if $|x_0^j| > 2^{-9}G\alpha\sqrt{n}$, then

$$\mathbb{E} [x_0^{j+1}] \geq x_0^j - 2^{-11}|x_0^j|L\alpha n.$$

Next, we aim to use the two lemmas above to get lower bounds on unconditioned expectation $\mathbb{E}[|x_0^{j+1}|]$. For this, we consider the following two cases.

Case 1: If $\mathbb{P}(|x_0^j| > 2^{-9}G\alpha\sqrt{n})\mathbb{E} [|x_0^j| | |x_0^j| > 2^{-9}G\alpha\sqrt{n}] > 2^{-10}G\alpha\sqrt{n}$. Then decomposing the expectation into conditional expectations,

$$\begin{aligned} \mathbb{E} [|x_0^j|] &= \mathbb{P} (|x_0^j| > 2^{-9}G\alpha\sqrt{n}) \mathbb{E} [|x_0^j| | |x_0^j| > 2^{-9}G\alpha\sqrt{n}] \\ &\quad + \mathbb{P} (|x_0^j| \leq 2^{-9}G\alpha\sqrt{n}) \mathbb{E} [|x_0^j| | |x_0^j| \leq 2^{-9}G\alpha\sqrt{n}] \\ &\geq \mathbb{P} (|x_0^j| > 2^{-9}G\alpha\sqrt{n}) \mathbb{E} [|x_0^j| | |x_0^j| > 2^{-9}G\alpha\sqrt{n}] \\ &> 2^{-10}G\alpha\sqrt{n}. \end{aligned} \tag{A.11}$$

Case 2: Otherwise, $\mathbb{P}(|x_0^j| > 2^{-9}G\alpha\sqrt{n})\mathbb{E} [|x_0^j| | |x_0^j| > 2^{-9}G\alpha\sqrt{n}] \leq 2^{-10}G\alpha\sqrt{n}$. Again

decomposing the expectation into conditional expectations,

$$\begin{aligned}
\mathbb{E}[x_0^{j+1}] &= \mathbb{P}(|x_0^j| \leq 2^{-9}G\alpha\sqrt{n}) \mathbb{E}[x_0^{j+1} | |x_0^j| \leq 2^{-9}G\alpha\sqrt{n}] \\
&\quad + \mathbb{P}(|x_0^j| > 2^{-9}G\alpha\sqrt{n}) \mathbb{E}[x_0^{j+1} | |x_0^j| > 2^{-9}G\alpha\sqrt{n}] \\
&\geq \mathbb{P}(|x_0^j| \leq 2^{-9}G\alpha\sqrt{n}) \left(\mathbb{E}[x_0^j | |x_0^j| \leq 2^{-9}G\alpha\sqrt{n}] + 2^{-12}LG\alpha^2n\sqrt{n} \right) \\
&\quad + \mathbb{P}(|x_0^j| > 2^{-9}G\alpha\sqrt{n}) \left(\mathbb{E}[x_0^j | |x_0^j| > 2^{-9}G\alpha\sqrt{n}] \right. \\
&\quad \left. - 2^{-11}L\mathbb{E}[|x_0^j| | |x_0^j| > 2^{-9}G\alpha\sqrt{n}] \alpha n \right) \\
&\hspace{15em} [\text{Using Lemma A.13 and A.14}] \\
&= \mathbb{E}[x_0^j] + \mathbb{P}(|x_0^j| \leq 2^{-9}G\alpha\sqrt{n}) 2^{-12}LG\alpha^2n\sqrt{n} \\
&\quad - \mathbb{P}(|x_0^j| > 2^{-9}G\alpha\sqrt{n}) \mathbb{E}[|x_0^j| | |x_0^j| > 2^{-9}G\alpha\sqrt{n}] 2^{-11}L\alpha n. \tag{A.12}
\end{aligned}$$

In the last step above, we gathered the sum of conditional expectations back into an unconditional expectation.

By assumption of this case,

$$\mathbb{P}(|x_0^j| > 2^{-9}G\alpha\sqrt{n}) \mathbb{E}[|x_0^j| | |x_0^j| > 2^{-9}G\alpha\sqrt{n}] \leq 2^{-10}G\alpha\sqrt{n}.$$

This implies that $\mathbb{P}(|x_0^j| > 2^{-9}G\alpha\sqrt{n}) \leq \frac{1}{2}$ and thus, $\mathbb{P}(|x_0^j| \leq 2^{-9}G\alpha\sqrt{n}) \geq \frac{1}{2}$. Using this in Ineq. (A.12),

$$\begin{aligned}
\mathbb{E}[x_0^{j+1}] &\geq \mathbb{E}[x_0^j] + \frac{1}{2}2^{-12}LG\alpha^2n\sqrt{n} \\
&\quad - \mathbb{P}(|x_0^j| > 2^{-9}G\alpha\sqrt{n}) \mathbb{E}[|x_0^j| | |x_0^j| > 2^{-9}G\alpha\sqrt{n}] 2^{-11}L\alpha n \\
&\geq \mathbb{E}[x_0^j] + 2^{-13}LG\alpha^2n\sqrt{n} - (2^{-10}G\alpha\sqrt{n}) 2^{-11}L\alpha n \\
&\hspace{15em} [\text{By assumption of this case}] \\
&\geq \mathbb{E}[x_0^j] + 2^{-14}LG\alpha^2n\sqrt{n}. \tag{A.13}
\end{aligned}$$

What we have shown using the two cases is the following: If for some epoch j , $\mathbb{E}[|x_0^j|] \leq 2^{-10}G\alpha\sqrt{n}$ then looking at Ineq. (A.11) tells us that we are in Case 2. Then, $\mathbb{E}[x_0^{j+1}] \geq \mathbb{E}[x_0^j] + 2^{-14}LG\alpha^2n\sqrt{n}$, that is $\mathbb{E}[x_0^{j+1}]$ increases by $\Omega(\alpha^2n\sqrt{n})$. This shows that if we initialize x_0^1 at 0, then until $\mathbb{E}[|x_0^j|] > 2^{-10}G\alpha\sqrt{n}$, the expected error will keep increasing at rate $\Omega(\alpha^2n\sqrt{n})$. Thus given the step size regime considered in this theorem, there is some epoch where error reaches $\Omega(\alpha\sqrt{n})$, which is the desired lower bound. However, what we want to show is that at the end of K epoch, the error is still $\Omega(\alpha\sqrt{n})$. We will prove this next.

We initialize $x_0^1 = 0$ and we run K epochs. Then, because $\mathbb{E}[|x_0^j|] \geq \mathbb{E}[x_0^j]$, we have

shown in the previous paragraph that $\mathbb{E}[|x_0^j|] \geq \min\{2^{-10}, 2^{-14}L\alpha nK\} G\alpha\sqrt{n}$ for some $0 \leq j \leq K$. Now, for our given range of α , we know that $L\alpha nK$ is greater than L , which in turn is greater than 2^{17} . Thus, $\mathbb{E}[|x_0^j|] \geq 2^{-10}G\alpha\sqrt{n}$. To complete the proof, next we prove that once $\mathbb{E}[|x_0^j|] \geq 2^{-10}G\alpha\sqrt{n}$, then $\mathbb{E}[|x_0^t|]$ remains above $C_l G\alpha\sqrt{n}$ for $t > j$ and some universal constant C_l . The strategy is that we will show that if $\mathbb{E}[|x_0^j|]$ starts falling below $2^{-10}G\alpha\sqrt{n}$, then $\mathbb{E}[x_0^j]$ starts increasing. We also have Lemma A.15 (given below) that says that $\mathbb{E}[x_0^j] \geq 0$ always; and the trivial fact that $\mathbb{E}[|x_0^j|] \geq \mathbb{E}[x_0^j]$ always. All these together and some simple arithmetic will give a bound on how much $\mathbb{E}[|x_0^j|]$ can decrease.

Lemma A.15. *If $\alpha \leq 1/L$, then $\forall i, j : \mathbb{E}[x_i^j] \geq 0$.*

We formalize the argument of the previous paragraph next. Let j be such that $\mathbb{E}[|x_0^j|] \geq 2^{-10}G\alpha\sqrt{n}$ and $\mathbb{E}[|x_0^{j+1}|] < 2^{-10}G\alpha\sqrt{n}$. Then,

$$\begin{aligned} \mathbb{E}[|x_0^{j+1}|] &\geq \mathbb{E}[|x_0^j|] (1 - 2L\alpha n) - \sqrt{5}LG\alpha^2 n\sqrt{n} && \text{[Using Lemma A.14]} \\ &\geq \frac{1}{2}\mathbb{E}[|x_0^j|] - \sqrt{5}LG\alpha^2 n\sqrt{n} && \text{[Since } \alpha \leq \frac{1}{4nL}\text{]} \\ &\geq 2^{-11}G\alpha\sqrt{n} - \sqrt{5}LG\alpha^2 n\sqrt{n} \\ &\geq 2^{-12}G\alpha\sqrt{n}. && \text{[Since } \alpha \leq \frac{2^{-14}}{nL}\text{]} \end{aligned}$$

For subsequent epochs $l > j$, we want to show that $\mathbb{E}[|x_0^l|]$ doesn't fall below $\Omega(\alpha\sqrt{n})$. Thus assume that for $l > j$, $\mathbb{E}[|x_0^l|] < 2^{-10}G\alpha\sqrt{n}$, because otherwise $\mathbb{E}[|x_0^l|] = \Omega(\alpha\sqrt{n})$.

Because $\mathbb{E}[|x_0^l|] < 2^{-10}G\alpha\sqrt{n}$, then using Ineq. (A.11), we can infer that we are in Case 2 in the epoch l . Therefore, Ineq. (A.13) implies that for each such epoch l , $\mathbb{E}[x_0^{l+1}]$ increases by at least $2^{-14}LG\alpha^2 n\sqrt{n}$ per epoch; whereas Lemma A.14 says that $\mathbb{E}[|x_0^{l+1}|]$ can decrease by at most $2L\mathbb{E}[|x_0^l|]\alpha n + \sqrt{5}LG\alpha^2 n\sqrt{n} \leq (2^{-11} + \sqrt{5})LG\alpha^2 n\sqrt{n}$ per epoch. Further, we have the two facts that $\forall l : \mathbb{E}[x_0^l] \geq 0$ (Lemma A.15) and $\mathbb{E}[x_0^l] \leq \mathbb{E}[|x_0^l|]$. Combining all these and using simple arithmetic gives that $\mathbb{E}[|x_0^l|]$ can decrease to at most

$$2^{-12}G\alpha\sqrt{n} \left(\frac{2^{-14}LG\alpha^2 n\sqrt{n}}{2^{-14}LG\alpha^2 n\sqrt{n} + (2^{-11} + \sqrt{5})LG\alpha^2 n\sqrt{n}} \right) \geq 2^{-28}G\alpha\sqrt{n}.$$

After this $\mathbb{E}[|x_0^l|]$ will have to keep increasing because $\mathbb{E}[x_0^l]$ will keep increasing till we enter Case 1, and then this cycle may repeat, but we have shown that regardless, $\mathbb{E}[|x_0^l|]$ always remains above $2^{-28}G\alpha\sqrt{n}$.

Finally, given $\alpha \geq \frac{1}{nK}$, we get that $\mathbb{E}[|x_0^K|] \geq 2^{-28}G\alpha\sqrt{n} \geq \frac{2^{-28}G}{\sqrt{nK}}$. Applying Jensen's inequality on this gives $\mathbb{E}[|x_0^K|^2] \geq \frac{2^{-56}G^2}{nK^2} = \frac{2^{-56}G^2 n}{T^2}$.

□

A.2.1 Proof of Lemma A.13

The gradient computed at x_i^j can be written as $(\mathbb{1}_{x_i^j \leq 0}L + \mathbb{1}_{x_i^j > 0}1)x_i^j + \frac{G}{2}\sigma_i^j$. Then, $x_0^{j+1} - x_0^j$ is just the sum of the gradient steps taken through the epoch.

$$\begin{aligned}
\mathbb{E}[x_0^{j+1}] &= x_0^j - \alpha \frac{G}{2} \sum_{i=1}^n \sigma_i^j - \alpha \sum_{i=0}^n \mathbb{E}[(\mathbb{1}_{x_i^j \leq 0}L + \mathbb{1}_{x_i^j > 0}1)x_i^j] \\
&= x_0^j - \alpha \sum_{i=1}^n \mathbb{E}[(\mathbb{1}_{x_i^j \leq 0}L + \mathbb{1}_{x_i^j > 0}1)x_i^j] \quad [\text{Since } \sum_{i=1}^n \sigma_i^j = 0] \\
&= x_0^j - \alpha \sum_{i=n/4}^{n/2} \mathbb{E}[(\mathbb{1}_{x_i^j \leq 0}L + \mathbb{1}_{x_i^j > 0}1)x_i^j] - \alpha \sum_{i \notin [\frac{n}{4}, \frac{n}{2}]} \mathbb{E}[(\mathbb{1}_{x_i^j \leq 0}L + \mathbb{1}_{x_i^j > 0}1)x_i^j] \\
&= x_0^j - \alpha \sum_{i=n/4}^{n/2} \mathbb{P}\left(\sum_{p=1}^i \sigma_p^j > 0\right) \mathbb{E}\left[(\mathbb{1}_{x_i^j \leq 0}L + \mathbb{1}_{x_i^j > 0}1)x_i^j \middle| \sum_{p=1}^i \sigma_p^j > 0\right] \\
&\quad - \alpha \sum_{i=n/4}^{n/2} \mathbb{P}\left(\sum_{p=1}^i \sigma_p^j \leq 0\right) \mathbb{E}\left[(\mathbb{1}_{x_i^j \leq 0}L + \mathbb{1}_{x_i^j > 0}1)x_i^j \middle| \sum_{p=1}^i \sigma_p^j \leq 0\right] \\
&\quad - \alpha \sum_{i \notin [\frac{n}{4}, \frac{n}{2}]} \mathbb{E}[(\mathbb{1}_{x_i^j \leq 0}L + \mathbb{1}_{x_i^j > 0}1)x_i^j]. \tag{A.14}
\end{aligned}$$

We decomposed the expectation into the sum of conditional expectations to achieve the last equality.

The following inequalities help us bound $\mathbb{E}[(\mathbb{1}_{x_i^j \leq 0}L + \mathbb{1}_{x_i^j > 0}1)x_i^j]$ in Eq. (A.14) with simple expressions. Let r be any random variable. Then,

$$\begin{aligned}
\mathbb{E}[(\mathbb{1}_{r \leq 0}L + \mathbb{1}_{r > 0}1)r] &= L\mathbb{P}(r < 0)\mathbb{E}[r | r < 0] + \mathbb{P}(r \geq 0)\mathbb{E}[r | r \geq 0] \\
&= L(\mathbb{P}(r < 0)\mathbb{E}[r | r < 0] + \mathbb{P}(r \geq 0)\mathbb{E}[r | r \geq 0]) \\
&\quad + (1 - L)\mathbb{P}(r \geq 0)\mathbb{E}[r | r \geq 0] \\
&\leq L(\mathbb{P}(r < 0)\mathbb{E}[r | r < 0] + \mathbb{P}(r \geq 0)\mathbb{E}[r | r \geq 0]) \quad [\text{Since } L \geq 1] \\
&\leq L\mathbb{E}[r] \tag{A.15}
\end{aligned}$$

and

$$\begin{aligned}
\mathbb{E}[(\mathbb{1}_{r \leq 0}L + \mathbb{1}_{r > 0}1)r] &= L\mathbb{P}(r < 0)\mathbb{E}[r | r < 0] + \mathbb{P}(r \geq 0)\mathbb{E}[r | r \geq 0] \\
&= 1(\mathbb{P}(r < 0)\mathbb{E}[r | r < 0] + \mathbb{P}(r \geq 0)\mathbb{E}[r | r \geq 0]) \\
&\quad + (L - 1)\mathbb{P}(r < 0)\mathbb{E}[r | r < 0] \\
&\leq 1(\mathbb{P}(r < 0)\mathbb{E}[r | r < 0] + \mathbb{P}(r \geq 0)\mathbb{E}[r | r \geq 0]) \quad [\text{Since } L \geq 1] \\
&\leq \mathbb{E}[r]. \tag{A.16}
\end{aligned}$$

We also have the following lemmas and corollary that along with the two inequalities above, help lower bound the RHS of Eq. (A.14).

Lemma A.16. *If $|x_0^j| \leq \sqrt{5n}G\alpha$ and $\alpha \leq \frac{2^{-11}}{nL}$, then for $n/4 \leq i \leq n/2$, we have*

$$\mathbb{E} \left[x_i^j - x_0^j \middle| \sum_{p=1}^i \sigma_p^j > 0 \right] \leq -\frac{1}{128}G\sqrt{i}\alpha.$$

Lemma A.17. *If $n \geq 256$ is a multiple of 4 and $i \leq n/2$, then*

$$\frac{\sqrt{i}}{32} \leq \mathbb{E} \left[\left| \sum_{p=1}^i \sigma_p^j \right| \right] \leq \sqrt{i}.$$

Further, for any $i \in [n/4, n/2]$,

$$\mathbb{P} \left(\sum_{p=1}^i \sigma_p^j < 0 \right) \geq \frac{1}{4} \quad \text{and} \quad \mathbb{P} \left(\sum_{p=1}^i \sigma_p^j > 0 \right) \geq \frac{1}{4}.$$

Corollary A.18. *If $\alpha \leq 2/L$ and $i \in [\frac{n}{4}, \frac{n}{2}]$ then for any $h \in [1, n]$,*

$$\begin{aligned} \mathbb{E} \left[|x_h^j - x_0^j| \middle| \sum_{p=1}^i \sigma_p^j \leq 0 \right] &\leq 4 \left(\sqrt{5h}G\alpha + |x_0^j|\sqrt{h\alpha L} \right), \text{ and} \\ \mathbb{E} \left[|x_h^j - x_0^j| \middle| \sum_{p=1}^i \sigma_p^j > 0 \right] &\leq 4 \left(\sqrt{5h}G\alpha + |x_0^j|\sqrt{h\alpha L} \right). \end{aligned}$$

Now, we handle each of the three terms in Eq. (A.14) individually.

- If $i \in [n/4, n/2]$ and $\left(\sum_{p=1}^i \sigma_p^j \right) > 0$:

$$\begin{aligned} \mathbb{E} \left[(\mathbb{1}_{x_i^j \leq 0}L + \mathbb{1}_{x_i^j > 0}1)x_i^j \middle| \sum_{p=1}^i \sigma_p^j > 0 \right] &\leq L\mathbb{E} \left[x_i^j \middle| \sum_{p=1}^i \sigma_p^j > 0 \right] \\ &\quad \text{[Using Ineq. (A.15) with } r = (x_i^j \mid \sum_{p=1}^i \sigma_p^j > 0)] \\ &= L\mathbb{E} \left[x_i^j - x_0^j \middle| \sum_{p=1}^i \sigma_p^j > 0 \right] + Lx_0^j \\ &\leq -\frac{L}{128}\sqrt{\frac{n}{4}}G\alpha + \frac{L}{512}G\sqrt{n}\alpha \\ &\quad \text{[Using Lemma A.16]} \end{aligned}$$

$$\leq -\frac{L}{512}G\sqrt{n}\alpha. \quad (\text{A.17})$$

- If $i \in [n/4, n/2]$ and $(\sum_{p=1}^i \sigma_p^j) \leq 0$:

$$\begin{aligned} \mathbb{E} \left[(\mathbb{1}_{x_i^j \leq 0} L + \mathbb{1}_{x_i^j > 0} 1) x_i^j \left| \sum_{p=1}^i \sigma_p^j \leq 0 \right. \right] &\leq \mathbb{E} \left[x_i^j \left| \sum_{p=1}^i \sigma_p^j \leq 0 \right. \right] \\ &\quad [\text{Using Ineq. (A.16) with } r = (x_i^j \mid \sum_{p=1}^i \sigma_p^j \leq 0)] \\ &\leq \mathbb{E} \left[x_i^j - x_0^j \left| \sum_{p=1}^i \sigma_p^j \leq 0 \right. \right] + |x_0^j| \\ &\leq 4 \left(\sqrt{5i}G\alpha + |x_0^j| \sqrt{2i\alpha L} \right) + |x_0^j| \\ &\quad [\text{Using Corollary A.18 with } h = i.] \\ &\leq 4 \left(\sqrt{5i}G\alpha + |x_0^j| \right) + |x_0^j| \quad [\text{Since } \alpha \leq 1/nL] \\ &\leq 4 \left(\sqrt{5i}G\alpha + 2^{-9}G\alpha\sqrt{n} \right) + 2^{-9}G\alpha\sqrt{n} \\ &\quad [\text{Since } |x_0^j| \leq 2^{-9}G\alpha\sqrt{n} \text{ by assumption}] \\ &\leq 8G\sqrt{n}\alpha. \quad (\text{A.18}) \end{aligned}$$

- If $i \notin [n/4, n/2]$:

$$\begin{aligned} \mathbb{E} \left[(\mathbb{1}_{x_i^j \leq 0} L + \mathbb{1}_{x_i^j > 0} 1) x_i^j \right] &\leq \mathbb{E} [x_i^j] \quad [\text{Using Ineq. (A.16) with } r = x_i^j] \\ &\leq \mathbb{E} [x_i^j - x_0^j] + |x_0^j| \\ &\leq \left(\sqrt{5i}G\alpha + |x_0^j| \sqrt{2i\alpha L} \right) + |x_0^j| \\ &\quad [\text{Using Lemma A.6 with } x^* = 0] \\ &\leq \left(\sqrt{5i}G\alpha + |x_0^j| \sqrt{2} \right) + |x_0^j| \quad [\text{Since } \alpha \leq 1/nL] \\ &\leq 8G\sqrt{n}\alpha. \quad (\text{A.19}) \end{aligned}$$

Continuing on from Eq. (A.14) and substituting Ineq. (A.17), Ineq. (A.18) and Ineq. (A.19):

$$\begin{aligned} \mathbb{E} [x_0^{j+1}] &= x_0^j - \alpha \sum_{i=n/4}^{n/2} \mathbb{P} \left(\sum_{p=1}^i \sigma_p^j > 0 \right) \mathbb{E} \left[(\mathbb{1}_{x_i^j \leq 0} L + \mathbb{1}_{x_i^j > 0} 1) x_i^j \left| \sum_{p=1}^i \sigma_p^j > 0 \right. \right] \\ &\quad - \alpha \sum_{i=n/4}^{n/2} \mathbb{P} \left(\sum_{p=1}^i \sigma_p^j \leq 0 \right) \mathbb{E} \left[(\mathbb{1}_{x_i^j \leq 0} L + \mathbb{1}_{x_i^j > 0} 1) x_i^j \left| \sum_{p=1}^i \sigma_p^j \leq 0 \right. \right] \end{aligned}$$

$$\begin{aligned}
& -\alpha \sum_{i \notin [\frac{n}{4}, \frac{n}{2}]} \mathbb{E} \left[(\mathbb{1}_{x_i^j \leq 0} L + \mathbb{1}_{x_i^j > 0} 1) x_i^j \right] \\
& \geq x_0^j - \alpha \sum_{i=n/4}^{n/2} \mathbb{P} \left(\sum_{p=1}^i \sigma_p^j > 0 \right) \left(-\frac{L}{512} G \sqrt{n} \alpha \right) \\
& \quad - \alpha \sum_{i=n/4}^{n/2} \mathbb{P} \left(\sum_{p=1}^i \sigma_p^j \leq 0 \right) (8G \sqrt{n} \alpha) - \alpha \sum_{i \notin [\frac{n}{4}, \frac{n}{2}]} (8G \sqrt{n} \alpha) \\
& \hspace{15em} [\text{Using Ineq. (A.17), (A.18) and (A.19)}] \\
& \geq x_0^j - \alpha \sum_{i=n/4}^{n/2} \mathbb{P} \left(\sum_{p=1}^i \sigma_p^j > 0 \right) \left(-\frac{L}{512} G \sqrt{n} \alpha \right) - \alpha \sum_{i=1}^n (8G \sqrt{n} \alpha) \\
& \geq x_0^j + \alpha \frac{n}{4} \frac{1}{4} \left(\frac{L}{512} G \sqrt{n} \alpha \right) - \alpha n (8G \sqrt{n} \alpha) \hspace{10em} [\text{Using Lemma A.17}] \\
& \geq x_0^j + 2^{-12} L G \alpha^2 n \sqrt{n}. \hspace{15em} [\text{Since } L \geq 2^{17}]
\end{aligned}$$

A.2.2 Proof of Lemma A.14

We will start off by bounding the difference of iterates between the start of two epochs. The gradient computed at x_i^j can be written as $(\mathbb{1}_{x_i^j \leq 0} L + \mathbb{1}_{x_i^j > 0} 1) x_i^j + \frac{G}{2} \sigma_i^j$. Then, $x_0^{j+1} - x_0^j$ is just the sum of the gradient steps taken through the epoch.

$$\begin{aligned}
\mathbb{E} [x_0^{j+1} - x_0^j] &= \mathbb{E} \left[-\alpha \frac{G}{2} \sum_{i=1}^n \sigma_i^j - \alpha \sum_{i=0}^n (\mathbb{1}_{x_i^j \leq 0} L + \mathbb{1}_{x_i^j > 0} 1) x_i^j \right] \\
&= \mathbb{E} \left[-\alpha \sum_{i=0}^n (\mathbb{1}_{x_i^j \leq 0} L + \mathbb{1}_{x_i^j > 0} 1) x_i^j \right] \hspace{10em} [\text{Since } \sum_{i=1}^n \sigma_i^j = 0] \\
&\leq \alpha \mathbb{E} \left[\sum_{i=0}^n (\mathbb{1}_{x_i^j \leq 0} L + \mathbb{1}_{x_i^j > 0} 1) |x_i^j| \right] \\
&\leq \alpha L \mathbb{E} \left[\sum_{i=0}^n |x_i^j| \right] \\
&\leq \alpha L \mathbb{E} \left[\sum_{i=0}^n (|x_0^j - x_i^j| + |x_0^j|) \right] \\
&\leq \alpha L \mathbb{E} \left[\sum_{i=0}^n (\sqrt{i \alpha} L |x_0^j| + \sqrt{5 i \alpha} G + |x_0^j|) \right] \hspace{10em} [\text{Using Lemma A.6}] \\
&\leq \alpha L \mathbb{E} \left[\sum_{i=0}^n (\sqrt{5 i \alpha} G + 2 |x_0^j|) \right] \hspace{10em} [\text{Since } \alpha \leq 1/nL] \\
&\leq \alpha L n (\sqrt{5 n \alpha} G + 2 |x_0^j|).
\end{aligned}$$

Thus, we have shown the following:

$$\mathbb{E} \left[|x_0^{j+1} - x_0^j| \right] \leq 2L|x_0^j|\alpha n + LG\alpha^2 n\sqrt{5n}. \quad (\text{A.20})$$

Then,

$$\begin{aligned} \mathbb{E} \left[|x_0^{j+1}| \right] &= \mathbb{E} \left[|x_0^j + x_0^{j+1} - x_0^j| \right] \\ &\geq |x_0^j| - \mathbb{E} \left[|x_0^{j+1} - x_0^j| \right] \\ &\geq |x_0^j|(1 - 2L\alpha n) - LG\alpha^2 n\sqrt{5n}. \end{aligned} \quad [\text{Using Ineq. (A.20)}]$$

This proves the first inequality of the lemma. For the second inequality,

$$\begin{aligned} \mathbb{E} \left[x_0^{j+1} \middle| |x_0^j| > \frac{G\alpha\sqrt{n}}{512} \right] &= \mathbb{E} \left[x_0^j + x_0^{j+1} - x_0^j \middle| |x_0^j| > 2^{-9}G\alpha\sqrt{n} \right] \\ &\geq x_0^j - \mathbb{E} \left[|x_0^{j+1} - x_0^j| \middle| |x_0^j| > 2^{-9}G\alpha\sqrt{n} \right] \\ &\geq x_0^j - (2|x_0^j|L\alpha n + LG\alpha^2 n\sqrt{5n}) \quad [\text{Using Ineq. (A.20)}] \\ &\geq x_0^j - (2L|x_0^j|\alpha n + 512\sqrt{5}L|x_0^j|\alpha n) \quad [\text{Since } |x_0^j| > \frac{G\alpha\sqrt{n}}{512}] \\ &\geq x_0^j - 2^{-11}L|x_0^j|\alpha n. \end{aligned}$$

A.2.3 Proof of Lemma A.15

Let us denote the iterate at iteration t by $x_{t,L}$, where the L in subscript is to show dependence on L . Now, consider the problem setup when $L = 1$. In that case, the two kinds of functions are simply quadratics instead of piecewise quadratics. The two kinds of functions in this case are $f_1(x) = \frac{1}{2}x^2 + \frac{G}{2}x$ and $f_2(x) = \frac{1}{2}x^2 - \frac{G}{2}x$. Then, due to symmetry of the function, the expected value of iterate at any iteration t is 0: $\mathbb{E}[x_{t,1}] = 0$.

Now, take the problem setup for the case $L \geq 1$, where we denote the iterates by $x_{t,L}$. We couple the two iterates $x_{t,1}$ and $x_{t,L}$ such that both $x_{t,L}$ and $x_{t,1}$ are created using the exact same random permutations of the functions but $x_{t,1}$ had $L = 1$ and $x_{t,L}$ had $L \geq 1$. Then, we show that $x_{t,L} \geq x_{t,1}$ and since $\mathbb{E}[x_{t,1}] = 0$, we get that $\mathbb{E}[x_{t,L}] \geq 0$.

We prove this by induction.

- Base case: Both are initialized at 0, and thus $x_{0,1} = x_{0,L} = 0$.
- Inductive case (assume $x_{i,L} \geq x_{i,1}$): We break the analysis into the following three cases:

1. **Case:** If $x_{i,1} \leq x_{i,L} \leq 0$. Then note that regardless of the choice of the functions, the contribution of the linear gradients in the gradients would be the same for both $x_{i+1,1}$ and $x_{i+1,L}$ (since we use the exact same permutation of functions for both). Hence,

$$\begin{aligned} x_{i+1,L} - x_{i+1,1} &= (1 - \alpha L)x_{i,L} - (1 - \alpha)x_{i,1} \\ &\geq (1 - \alpha L)x_{i,L} - (1 - L\alpha)x_{i,1} \\ &\geq 0. \end{aligned} \quad [\text{Since } \alpha \leq 1/L]$$

2. **Case:** If $0 \leq x_{i,1} \leq x_{i,L}$. Again, regardless of the choice of the functions, the contribution of the linear gradients in the gradients would be the same for both $x_{i+1,1}$ and $x_{i+1,L}$ (since we use the exact same permutation of functions for both). Hence,

$$\begin{aligned} x_{i+1,L} - x_{i+1,1} &= (1 - \alpha)x_{i,L} - (1 - \alpha)x_{i,1} \\ &\geq 0. \end{aligned} \quad [\text{Since } \alpha \leq 1/L]$$

3. **Case:** If $x_{i,1} \leq 0 \leq x_{i,L}$. Again, regardless of the choice of the functions, the contribution of the linear gradients in the gradients would be the same for both $x_{i+1,1}$ and $x_{i+1,L}$ (since we use the exact same permutation of functions for both). Hence,

$$\begin{aligned} x_{i+1,L} - x_{i+1,1} &= (1 - \alpha)x_{i,L} - (1 - \alpha)x_{i,1} \\ &\geq 0. \end{aligned} \quad [\text{Since } \alpha \leq 1/L]$$

A.2.4 Proof of Lemma A.16

The gradient computed at x_i^j can be written as $(\mathbb{1}_{x_i^j \leq 0}L + \mathbb{1}_{x_i^j > 0}1)x_i^j + \frac{G}{2}\sigma_i^j$. Then,

$$\begin{aligned} \mathbb{E} \left[x_i^j - x_0^j \middle| \sum_{p=1}^i \sigma_p^j > 0 \right] &= \mathbb{E} \left[-\alpha \sum_{p=1}^i \nabla f_{\sigma_p^j}(x_{p-1}^j) \middle| \sum_{p=1}^i \sigma_p^j > 0 \right] \\ &= -\alpha \mathbb{E} \left[\sum_{p=1}^i \left((\mathbb{1}_{x_{p-1}^j \geq 0} + L\mathbb{1}_{x_{p-1}^j < 0})x_{p-1}^j + \frac{G}{2}\sigma_p^j \right) \middle| \sum_{p=1}^i \sigma_p^j > 0 \right] \end{aligned}$$

$$\begin{aligned}
&\leq -\alpha \mathbb{E} \left[- \left| \sum_{p=1}^i (\mathbb{1}_{x_{p-1}^j \geq 0} + L \mathbb{1}_{x_{p-1}^j < 0}) x_{p-1}^j \right| \right. \\
&\quad \left. + \sum_{p=1}^i \frac{G}{2} \sigma_p^j \left| \sum_{p=1}^i \sigma_p^j > 0 \right| \right] \\
&\leq -\alpha \mathbb{E} \left[-L \sum_{p=1}^i |x_{p-1}^j| + \frac{G}{2} \sum_{p=1}^i \sigma_p^j \left| \sum_{p=1}^i \sigma_p^j > 0 \right| \right] \\
&\leq -\alpha \mathbb{E} \left[\frac{G}{2} \left| \sum_{p=1}^i \sigma_p^j \right| - L \sum_{p=1}^i (|x_0^j - x_{p-1}^j| + |x_0^j|) \left| \sum_{p=1}^i \sigma_p^j > 0 \right| \right]. \tag{A.21}
\end{aligned}$$

We have the following helpful lemma which will help us control the first term in the sum above.

Lemma A.19. *If $n \geq 256$ is a multiple of 4 and $i \leq n/2$, then*

$$\frac{\sqrt{i}}{32} \leq \mathbb{E} \left[\left| \sum_{p=1}^i \sigma_p^j \right| \right] \leq \sqrt{i}.$$

Further, for any $i \in [n/4, n/2]$,

$$\mathbb{P} \left(\sum_{p=1}^i \sigma_p^j < 0 \right) \geq \frac{1}{4} \quad \text{and} \quad \mathbb{P} \left(\sum_{p=1}^i \sigma_p^j > 0 \right) \geq \frac{1}{4}.$$

We also use the following corollary (of Lemma A.17 and Lemma A.6).

Corollary A.20. *If $\alpha \leq 2/L$ and $i \in [\frac{n}{4}, \frac{n}{2}]$ then for any $h \in [1, n]$,*

$$\begin{aligned}
\mathbb{E} \left[|x_h^j - x_0^j| \left| \sum_{p=1}^i \sigma_p^j \leq 0 \right| \right] &\leq 4 \left(\sqrt{5h} G \alpha + |x_0^j| \sqrt{h \alpha L} \right), \text{ and} \\
\mathbb{E} \left[|x_h^j - x_0^j| \left| \sum_{p=1}^i \sigma_p^j > 0 \right| \right] &\leq 4 \left(\sqrt{5h} G \alpha + |x_0^j| \sqrt{h \alpha L} \right).
\end{aligned}$$

Continuing from (A.21), and using Lemma A.17 and Corollary A.18, we have that for $n/4 \leq i \leq n/2$

$$\mathbb{E} \left[|x_i^j - x_0^j| \left| \sum_{p=1}^i \sigma_p^j > 0 \right| \right] \leq -\frac{\sqrt{i} \alpha G}{64} + \alpha L i \left(4\sqrt{5i} G \alpha + |x_0^j| (1 + 4\sqrt{i \alpha L}) \right).$$

Thus, if $|x_0^j| \leq \sqrt{5n}G\alpha$ and $\alpha \leq \frac{2^{-11}}{nL}$, then for $n/4 \leq i \leq n/2$, we have

$$\mathbb{E} \left[x_i^j - x_0^j \middle| \sum_{p=1}^i \sigma_p^j > 0 \right] \leq -\frac{1}{128} \sqrt{i} \alpha G.$$

A.2.5 Proof of Lemma A.17

Proof. We skip the superscript in the notation because we'll be working inside an epoch.

Let $s_i := \sum_{p=1}^i \sigma_p$. First, we prove the upper bound. We have that

$$\begin{aligned} \mathbb{E}[|s_i|] &= \mathbb{E} \left[\left| \sum_{p=1}^i \sigma_p \right| \right] \\ &\leq \sqrt{\mathbb{E} \left[\left(\sum_{p=1}^i \sigma_p \right)^2 \right]} && \text{[Jensen's inequality]} \\ &= \sqrt{\sum_{p=1}^i \mathbb{E}[(\sigma_p)^2] + 2 \sum_{k < p \leq i} \mathbb{E}[\sigma_k \sigma_p]} \\ &= \sqrt{i + 2 \sum_{k < p \leq i} \mathbb{E}[\sigma_k \sigma_p]}. \end{aligned}$$

Now, $\mathbb{E}(\sigma_k \sigma_p) < 0$ because they are negatively correlated. Hence,

$$\mathbb{E}[|s_i|] \leq \sqrt{i}.$$

For the lower bound we start by decomposing the expectation into sum of conditional expectations:

$$\begin{aligned} \mathbb{E}[|s_i|] &= \mathbb{P}(s_{i-1}\sigma_i \geq 0) \mathbb{E} \left[|s_i| \middle| s_{i-1}\sigma_i \geq 0 \right] + \mathbb{P}(s_{i-1}\sigma_i < 0) \mathbb{E} \left[|s_i| \middle| s_{i-1}\sigma_i < 0 \right] \\ &= \mathbb{P}(s_{i-1}\sigma_i \geq 0) \mathbb{E} \left[|s_{i-1}| + 1 \middle| s_{i-1}\sigma_i \geq 0 \right] && \text{[Since } s_i = s_{i-1} + 1 \text{ if } s_{i-1}\sigma_i \geq 0\text{]} \\ &\quad + \mathbb{P}(s_{i-1}\sigma_i < 0) \mathbb{E} \left[|s_{i-1}| - 1 \middle| s_{i-1}\sigma_i < 0 \right] && \text{[Since } s_i = s_{i-1} - 1 \text{ if } s_{i-1}\sigma_i < 0\text{]} \\ &= \mathbb{P}(s_{i-1}\sigma_i \geq 0) \mathbb{E} \left[|s_{i-1}| \middle| s_{i-1}\sigma_i \geq 0 \right] + \mathbb{P}(s_{i-1}\sigma_i < 0) \mathbb{E} \left[|s_{i-1}| \middle| s_{i-1}\sigma_i < 0 \right] \\ &\quad + \mathbb{P}(s_{i-1}\sigma_i \geq 0) - \mathbb{P}(s_{i-1}\sigma_i < 0) \\ &= \mathbb{E}[|s_{i-1}|] + \mathbb{P}(s_{i-1}\sigma_i \geq 0) - \mathbb{P}(s_{i-1}\sigma_i < 0) \\ &= \mathbb{E}[|s_{i-1}|] + \mathbb{P}(s_{i-1} = 0) + \mathbb{P}(s_{i-1}\sigma_i > 0) - \mathbb{P}(s_{i-1}\sigma_i < 0). \end{aligned} \tag{A.22}$$

Intuitively, because $\mathbb{P}(s_{i-1}\sigma_i > 0) \approx \mathbb{P}(s_{i-1}\sigma_i < 0)$, then $\mathbb{E}[|s_i|] \approx \sum_{p=1}^i \mathbb{P}(s_i = 0)$. This is relatively easy to compute using combinatorics and gives that $\mathbb{E}[|s_i|] \approx \Omega(\sqrt{i})$. We do the exact calculations next.

Continuing on from Eq. (A.22) and decomposing the probabilities as sum of conditional probabilities,

$$\begin{aligned} \mathbb{E}[|s_i|] &= \mathbb{E}[|s_{i-1}|] + \mathbb{P}(s_{i-1} = 0) + \mathbb{P}(s_{i-1}\sigma_i > 0) - \mathbb{P}(s_{i-1}\sigma_i < 0) \\ &= \mathbb{E}[|s_{i-1}|] + \mathbb{P}(s_{i-1} = 0) + \sum_{p=1}^{i-1} \mathbb{P}(|s_{i-1}| = p) \mathbb{P}(s_{i-1}\sigma_i > 0 \mid |s_{i-1}| = p) \\ &\quad - \sum_{p=1}^{i-1} \mathbb{P}(|s_{i-1}| = p) \mathbb{P}(s_{i-1}\sigma_i < 0 \mid |s_{i-1}| = p). \end{aligned}$$

The term $\mathbb{P}(s_{i-1}\sigma_i > 0 \mid |s_{i-1}| = p)$ has a closed form solution: Assume WLOG that $s_i = 0$. Then, $\mathbb{P}(s_{i-1}\sigma_i > 0 \mid |s_{i-1}| = p) = \mathbb{P}(\sigma_i = +1 \mid s_{i-1} = p)$ is just the probability of sampling a $+1$, when out of $i-1$ previous samples there were p $+1$'s more than -1 's. This is just $\frac{(n-i-p+1)/2}{n-i+1}$. Similarly, we can handle the other cases. This gives,

$$\begin{aligned} \mathbb{E}[|s_i|] &= \mathbb{E}[|s_{i-1}|] + \mathbb{P}(s_{i-1} = 0) + \sum_{p=1}^{i-1} \mathbb{P}(|s_{i-1}| = p) \left(\frac{(n-i-p+1)/2}{n-i+1} \right) \\ &\quad - \sum_{p=1}^{i-1} \mathbb{P}(|s_{i-1}| = p) \left(\frac{(n-i+p+1)/2}{n-i+1} \right) \\ &= \mathbb{E}[|s_{i-1}|] + \mathbb{P}(s_{i-1} = 0) - \sum_{p=1}^{i-1} \mathbb{P}(|s_{i-1}| = p) \left(\frac{p}{n-i+1} \right) \\ &= \mathbb{E}[|s_{i-1}|] + \mathbb{P}(s_{i-1} = 0) - \mathbb{E}(|s_{i-1}|) \frac{1}{n-i+1} \\ &= \mathbb{E}[|s_{i-1}|] (1 - 1/(n-i+1)) + \mathbb{P}(s_{i-1} = 0) \\ &\geq \mathbb{E}[|s_{i-1}|] (1 - 2/n) + \mathbb{P}(s_{i-1} = 0) \quad [\text{Since } i \leq n/2] \\ &= \mathbb{E}[|s_{i-1}|] (1 - 2/n) + \mathbb{1}_{i-1 \text{ is even}} \mathbb{P}(s_{i-1} = 0). \quad [s_{i-1} \text{ can be 0 only if } i-1 \text{ is even.}] \end{aligned}$$

To compute $\mathbb{P}(s_{i-1} = 0)$, we first see that it is just the ratio of the number of ways of choosing $(i-1)/2$ positions for $+1$'s in the first $i-1$ positions and then choosing $(n-i+1)/2$ positions from the remaining $n-i+1$ positions for the remaining $+1$'s, to the total number of ways of choosing $n/2$ positions for $+1$'s from n positions. Thus,

$$\mathbb{P}(|s_{i-1}| = 0) = \frac{\binom{i-1}{(i-1)/2} \binom{n-i+1}{(n-i+1)/2}}{\binom{n}{n/2}} \quad (\text{A.23})$$

Using this equality and continuing on,

$$\begin{aligned}\mathbb{E}[|s_i|] &\geq \mathbb{E}[|s_{i-1}|](1 - 2/n) + \mathbb{1}_{i-1 \text{ is even}} \mathbb{P}(|s_{i-1}| = 0) \\ &= \mathbb{E}[|s_{i-1}|](1 - 2/n) + \mathbb{1}_{i-1 \text{ is even}} \frac{\binom{i-1}{(i-1)/2} \binom{n-i+1}{(n-i+1)/2}}{\binom{n}{n/2}}. \quad [\text{Using Eq. (A.23)}]\end{aligned}$$

To bound the combinatorial expression above, we use the following approximation by Mortici (2011, Theorem 1): $\sqrt{\pi(2k+0.33)} \left(\frac{k}{e}\right)^k \leq k! \leq \sqrt{\pi(2k+0.36)} \left(\frac{k}{e}\right)^k$. Using this and continuing on the sequence of inequalities,

$$\begin{aligned}\mathbb{E}[|s_i|] &\geq \mathbb{E}[|s_{i-1}|](1 - 2/n) + \mathbb{1}_{i-1 \text{ is even}} \frac{1}{16\sqrt{i-1}} \\ &= \mathbb{E}[|s_{i-2}|](1 - 2/n)^2 + \mathbb{1}_{i-2 \text{ is even}} \frac{1}{16\sqrt{i-2}}(1 - 2/n) + \mathbb{1}_{i-1 \text{ is even}} \frac{8\pi^2}{e^5} \sqrt{\frac{1}{i-1}} \\ &\geq \mathbb{E}[|s_{i-2}|](1 - 2/n)^2 + \frac{1}{16\sqrt{i}}(1 - 2/n) \\ &\geq \frac{(1 - 2/n)}{16\sqrt{i}} \sum_{p=0}^{\lfloor i/2 \rfloor - 1} (1 - 2/n)^{2p} \\ &= \frac{(1 - 2/n)}{16\sqrt{i}} \frac{1 - (1 - 2/n)^{2(\lfloor i/2 \rfloor)}}{\frac{4}{n} - \frac{4}{n^2}}.\end{aligned}$$

We will use the inequality : $\forall x \in [0, 1], m \geq 0 : (1 - x)^m \leq \frac{1}{1+mx}$ to upper bound the term $(1 - 2/n)^{\lfloor i/2 \rfloor}$. Thus, we get

$$\begin{aligned}\mathbb{E}[|s_i|] &\geq \frac{(1 - 2/n)}{16\sqrt{i}} \frac{1 - \frac{1}{1 + \frac{4}{n}\lfloor i/2 \rfloor}}{\frac{4}{n} - \frac{4}{n^2}} \\ &\geq \frac{(1 - 2/n)}{16\sqrt{i}} \frac{1 - \frac{1}{1 + \frac{i}{n}}}{\frac{4}{n}} \\ &= \frac{(1 - 2/n)}{16\sqrt{i}} \frac{in}{n+i} \\ &= \frac{(n-2)}{16} \frac{\sqrt{i}}{n+i} \\ &\geq \frac{(3n/4)}{16} \frac{\sqrt{i}}{3n/2} \\ &= \frac{\sqrt{i}}{32}.\end{aligned}$$

This proves the lower bound of the lemma.

Next we prove that $\mathbb{P}\left(\sum_{p=1}^i \sigma_p^j < 0\right) \geq \frac{1}{4}$ and $\mathbb{P}\left(\sum_{p=1}^i \sigma_p^j > 0\right) \geq \frac{1}{4}$. Firstly, we can see

that $\mathbb{P}\left(\sum_{p=1}^i \sigma_p^j < 0\right) = P\left(\sum_{p=1}^i \sigma_p^j > 0\right)$ by symmetry and hence it is sufficient to show $\mathbb{P}\left(\sum_{p=1}^i \sigma_p^j = 0\right) \leq 1/2$.

For odd i , $\mathbb{P}\left(\sum_{p=1}^i \sigma_p^j = 0\right) = 0$ trivially. Thus, we focus only on even i . Towards that end,

$$\begin{aligned}
\mathbb{P}\left(\sum_{p=1}^i \sigma_p^j = 0\right) &= \mathbb{P}(s_i = 0) \\
&= \frac{\binom{i}{i/2} \binom{n-i}{(n-i)/2}}{\binom{n}{n/2}} && \text{[Using Eq. (A.23)]} \\
&\leq \frac{4}{\sqrt{i}} && \text{[Using Mortici (2011, Theorem 1) mentioned earlier.]} \\
&\leq \frac{8}{\sqrt{n}} \\
&\leq \frac{1}{2}.
\end{aligned}$$

Thus, $\mathbb{P}\left(\sum_{p=1}^i \sigma_p^j < 0\right) = \mathbb{P}\left(\sum_{p=1}^i \sigma_p^j > 0\right) \geq 1/4$.

□

A.2.6 Proof of Corollary A.18

$$\begin{aligned}
\mathbb{E}[|x_h^j - x_0^j|] &= \mathbb{P}\left(\sum_{p=1}^i \sigma_p^i \leq 0\right) \mathbb{E}\left[|x_h^j - x_0^j| \middle| \sum_{p=1}^i \sigma_p^i \leq 0\right] \\
&\quad + \mathbb{P}\left(\sum_{p=1}^i \sigma_p^i > 0\right) \mathbb{E}\left[|x_h^j - x_0^j| \middle| \sum_{p=1}^i \sigma_p^i > 0\right] \\
&\geq \mathbb{P}\left(\sum_{p=1}^i \sigma_p^i \leq 0\right) \mathbb{E}\left[|x_h^j - x_0^j| \middle| \sum_{p=1}^i \sigma_p^i \leq 0\right].
\end{aligned}$$

Therefore,

$$\begin{aligned}
\mathbb{E}\left[|x_h^j - x_0^j| \middle| \sum_{p=1}^i \sigma_p^i \leq 0\right] &\leq \frac{\mathbb{E}[|x_h^j - x_0^j|]}{\mathbb{P}(\sum_{p=1}^i \sigma_p^i \leq 0)} \\
&\leq 4\mathbb{E}[|x_h^j - x_0^j|] && \text{[Using Lemma A.17]} \\
&\leq 4(\sqrt{5h}G\alpha + |x_0^j|\sqrt{h\alpha L}). && \text{[Using Lemma A.6]}
\end{aligned}$$

The other inequality can be proved similarly.

A.3 Proof of Corollary 4.13

Let $F_1(x)$ be the 1-Dimensional function from Theorem 4.12 with $L = 2^{17}$ and $F_2(x)$ be the 1-Dimensional function from Proposition 1 in Safran and Shamir (2020, p. 10-12) with $\lambda = 1$. In particular under this setting, the function from Proposition 1 in their paper is the following:

$$F_2(x) = \frac{1}{n} \sum_{i=1}^n f_{2,i}(x) = \frac{x^2}{2}, \text{ where}$$

$$\forall i \in [n] : f_{2,i}(x) = \begin{cases} \frac{x^2}{2} + \frac{Gx}{2}, & \text{if } x \geq 0 \\ \frac{x^2}{2} + \frac{Gx}{2}, & \text{if } x < 0. \end{cases}$$

In particular, it is the same function as Theorem 4.12 from this thesis if L was set to 1. Then, $F_1(x)$ and $F_2(x)$ satisfy the following properties (for each $j \in \{1, 2\}$):

1. $F_j(x) = \frac{1}{n} \sum_{i=1}^n f_{j,i}(x)$.
2. $F_j(x)$ satisfies Assumption 4.2, $\forall i, f_{j,i}(x)$ satisfy Assumptions 4.1 and 4.6; and that in the 1-Dimensional space, Assumptions 4.3 and 4.4 are satisfied. We prove these for our function $F_1(x)$, in the proof of Theorem 4.12. These can be similarly also proved for $F_2(x)$.
3. There is a step size range $A_j \subset \mathbb{R}$ and an initialization $x_{j,0}$ such that after K epochs of RR on $F_j(x)$ with any constant step size $\alpha \in A_j$,

$$\mathbb{E}[\|x_T - x_j^*\|^2] = \mathbb{E}[\|x_T\|^2] \geq C_l \frac{G^2 n}{T^2}, \quad (\text{A.24})$$

where $x_j^* = 0$ is the minimizer of $F_j(x)$ and C_l is a universal constant. When $L = 2^{17}$, we have shown that $A_1 = [\frac{1}{T}, \frac{C}{n}]$ for a universal constant C . The proof of Proposition 1 in Safran and Shamir (2020, p. 10-12) can be modified slightly so that F_2 satisfies Ineq. (A.24) with $A_2 = [0, \infty) \setminus A_1$.

For the rest of this proof, we will be working in a 2-Dimensional space. A 2-Dimensional vector will be represented as $x = [x^{(1)}, x^{(2)}]$. The superscript in this section **does not** denote the epoch, it denotes the co-ordinate.

$\forall i$, we define the 2-Dimensional functions $f_i(x) = f_{1,i}(x^{(1)}) + f_{2,i}(x^{(2)})$ and $F(x) = \frac{1}{n} \sum_{i=1}^n f_i(x)$. Then, $F(x)$ satisfies Assumption 4.2 and $\forall i, f_i(x)$ satisfy Assumptions 4.1 and 4.6. Let $x^* = [x_1^*, x_2^*] = [0, 0]$ denote the minimizer of $F(x)$. Consider the 2-Dimensional

square such which is defined by $\{x : \|x - x^*\|_\infty \leq D\}$. Then, in this domain, Assumptions 4.3 and 4.4 are satisfied with constants $D' := D\sqrt{2}$ and $G' := G\sqrt{2}$ respectively.

Note that the gradient of $f_i(x)$ with respect to $x^{(1)}$ is just $\nabla f_{1,i}(x^{(1)})$ and with respect to $x^{(2)}$ is just $\nabla f_{2,i}(x^{(2)})$. Thus, RR essentially operates independently along each of the two co-ordinates. Now, for any step length $\alpha \in \mathbb{R}^+$, we know that $\alpha \in A_1$ or $\alpha \in A_2$. Then using Ineq. (A.24), at least one of the following is true:

$$\mathbb{E}[\|x_T^{(1)} - (x^*)^{(1)}\|^2] = \mathbb{E}[\|x_T^{(1)} - x_1^*\|^2] \geq C_l \frac{G^2 n}{T^2}$$

or

$$\mathbb{E}[\|x_T^{(2)} - (x^*)^{(2)}\|^2] = \mathbb{E}[\|x_T^{(2)} - x_2^*\|^2] \geq C_l \frac{G^2 n}{T^2}.$$

Therefore, the overall error

$$\mathbb{E}[\|x_T - x^*\|^2] \geq C_l \frac{G^2 n}{T^2}.$$

A.4 Numerical results

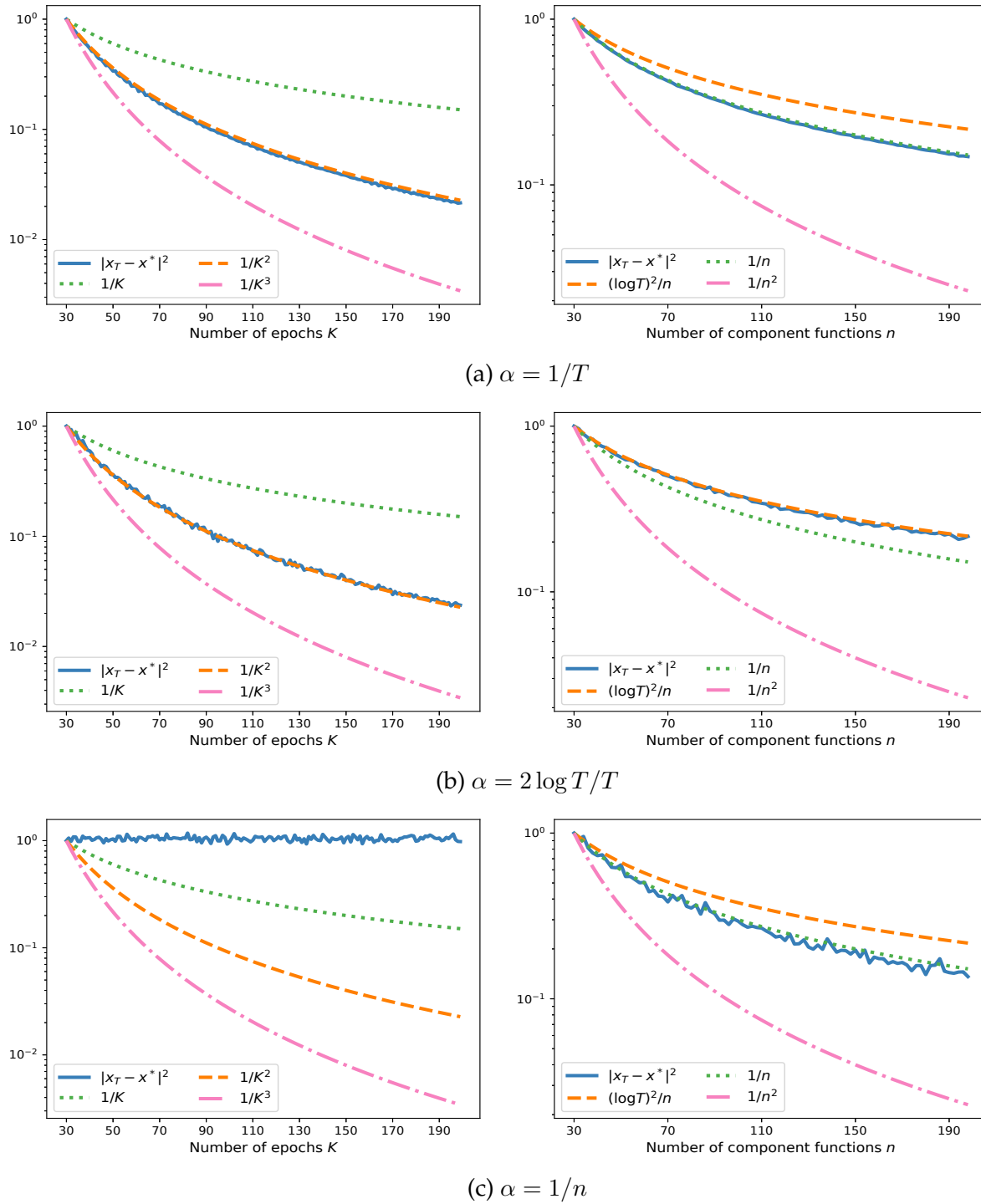


Figure A.4: Running RR with different step size regimes on the function F used in our lower bound, Theorem 4.12. The setup is the same as described in Subsection 4.5.1.

B APP: RANDOM RESHUFFLING IN DISTRIBUTED AND FEDERATED LEARNING

B.1 Comparisons with assumptions and rates in existing results

In this section, we compare our assumptions and convergence bounds against other existing results mentioned in the main text. Most existing results that study independent and unbiased gradient estimates state their assumptions in terms of the expectation over the randomness in the estimate; for such assumptions, we adapt them to our finite sum setting in order to make for easier comparison.

Heterogeneity assumptions. We start by discussing our definition of objective-wise heterogeneity (Assumption 5.3), namely that there exist $\tau \geq 0$ and $\rho \geq 1$ such that

$$\frac{1}{M} \sum_{m=1}^M \|\nabla F^m(\mathbf{x})\| \leq \tau + \rho \|\nabla F(\mathbf{x})\|, \text{ for all } \mathbf{x} \in \mathbb{R}^d. \quad (\text{B.1})$$

Perhaps the most relevant to this assumption is the (G, B) -BGD assumption that appears in Karimireddy et al. (2020): For all $\mathbf{x} \in \mathbb{R}^d$,

$$\frac{1}{M} \sum_{m=1}^M \|\nabla F^m(\mathbf{x})\|^2 \leq G^2 + B^2 \|\nabla F(\mathbf{x})\|^2. \quad (\text{B.2})$$

Note that thanks to Jensen's inequality and $a^2 + b^2 \leq (a + b)^2$ for $a, b \geq 0$, (B.2) implies

$$\begin{aligned} \left(\frac{1}{M} \sum_{m=1}^M \|\nabla F^m(\mathbf{x})\| \right)^2 &\leq \frac{1}{M} \sum_{m=1}^M \|\nabla F^m(\mathbf{x})\|^2 \leq G^2 + B^2 \|\nabla F(\mathbf{x})\|^2 \\ &\leq (G + B \|\nabla F(\mathbf{x})\|)^2, \end{aligned}$$

and hence (B.1) with $\tau = G$ and $\rho = B$. Therefore, our Assumption 5.3 is weaker than the (G, B) -BGD assumption. Several papers (Haddadpour and Mahdavi, 2019; Li et al., 2020b) use the same assumption (B.2), with $G = 0$; therefore, Assumption 5.3 also subsumes the heterogeneity assumption posed in these papers. Note that $G = 0$ implies that, the minima for F are also the minima for F^m , for every m , and hence $G = 0$ results in a weak form of heterogeneity.

Some papers (Yu et al., 2019; Li et al., 2020c; Qu et al., 2020) assume bounded local

gradients: for all $m \in [M]$ and $\mathbf{x} \in \mathbb{R}^d$,

$$\frac{1}{N} \sum_{i=1}^N \|\nabla f_i^m(\mathbf{x})\|^2 \leq G^2, \quad (\text{B.3})$$

and this in fact implies the $(G, 0)$ -BGD assumption (B.2). To see why, from Jensen's inequality

$$\|\nabla F^m(\mathbf{x})\|^2 := \left\| \frac{1}{N} \sum_{i=1}^N \nabla f_i^m(\mathbf{x}) \right\|^2 \leq \frac{1}{N} \sum_{i=1}^N \|\nabla f_i^m(\mathbf{x})\|^2 \leq G^2.$$

Therefore, (B.3) is a stronger assumption for objective-wise heterogeneity than Assumption 5.3.

In Theorem 3 of Woodworth et al. (2020b), the authors use the following assumption on heterogeneity: for all $\mathbf{x} \in \mathbb{R}^d$,

$$\frac{1}{M} \sum_{m=1}^M \|\nabla F^m(\mathbf{x}) - \nabla F(\mathbf{x})\|^2 \leq \bar{\zeta}^2. \quad (\text{B.4})$$

Noting that $\frac{1}{M} \sum_{m=1}^M \|\nabla F^m(\mathbf{x}) - \nabla F(\mathbf{x})\|^2 = \frac{1}{M} \sum_{m=1}^M \|\nabla F^m(\mathbf{x})\|^2 - \|\nabla F(\mathbf{x})\|^2$, we can see that (B.4) implies (B.2) and hence Assumption 5.2 (B.1), with $\tau = \bar{\zeta}$ and $\rho = 1$.

Indeed, there are also some results that make weaker heterogeneity assumptions than ours (B.1), by requiring bounded deviation only at the global optimum $\mathbf{x}^*$. Given the global optimum $\mathbf{x}^*$ of F , Koloskova et al. (2020) define

$$\bar{\zeta}_*^2 := \frac{1}{M} \sum_{m=1}^M \|\nabla F^m(\mathbf{x}^*)\|^2, \quad (\text{B.5})$$

and use this constant in their bounds. Khaled et al. (2020) also define a similar quantity that can capture heterogeneity, but does not provide a result on strongly convex functions in the heterogeneous setting. While assuming bounded $\bar{\zeta}_*^2$ (B.5) is weaker than Assumption 5.3 in the sense that only a bound at $\mathbf{x}^*$ is required, we note that these assumptions cannot be applied easily in nonconvex settings; in fact, for nonconvex (but not necessarily PL) functions, Koloskova et al. (2020) also use (B.2) as their heterogeneity assumption.

Intra-machine variance assumptions. We next consider our notion of intra-machine deviation (Assumption 5.2), namely that there exists $\nu \geq 0$ such that for all $m \in [M]$ and $i \in [N]$,

$$\|\nabla f_i^m(\mathbf{x}) - \nabla F^m(\mathbf{x})\| \leq \nu, \quad \text{for all } \mathbf{x} \in \mathbb{R}^d. \quad (\text{B.6})$$

In many existing results using independent and unbiased gradient estimates (Karimireddy et al., 2020; Yu et al., 2019; Li et al., 2020c; Qu et al., 2020; Woodworth et al., 2020a,b, 2021), the bounded variance assumption is adopted: For all $m \in [M]$ and $\mathbf{x} \in \mathbb{R}^d$,

$$\frac{1}{N} \sum_{i=1}^N \|\nabla f_i^m(\mathbf{x}) - \nabla F^m(\mathbf{x})\|^2 \leq \sigma^2. \quad (\text{B.7})$$

We note that the bounded local gradients assumption (B.3) also implies (B.7), by

$$\begin{aligned} \frac{1}{N} \sum_{i=1}^N \|\nabla f_i^m(\mathbf{x}) - \nabla F^m(\mathbf{x})\|^2 &\leq \frac{1}{N} \sum_{i=1}^N \|\nabla f_i^m(\mathbf{x}) - \nabla F^m(\mathbf{x})\|^2 + \|\nabla F^m(\mathbf{x})\|^2 \\ &= \frac{1}{N} \sum_{i=1}^N \|\nabla f_i^m(\mathbf{x})\|^2 \leq G^2. \end{aligned} \quad (\text{B.8})$$

Some other papers (Haddadpour et al., 2019; Haddadpour and Mahdavi, 2019; Spiridonoff et al., 2020) consider a generalized version of (B.7), namely that for all $m \in [M]$ and $\mathbf{x} \in \mathbb{R}^d$,

$$\frac{1}{N} \sum_{i=1}^N \|\nabla f_i^m(\mathbf{x}) - \nabla F^m(\mathbf{x})\|^2 \leq c \|\nabla F^m(\mathbf{x})\|^2 + \sigma^2, \quad (\text{B.9})$$

for $c, \sigma \geq 0$. There are other papers (Koloskova et al., 2020; Khaled et al., 2020) that use intra-machine variance at the global minimum $\mathbf{x}^*$ in their bounds. Koloskova et al. (2020) define

$$\bar{\sigma}_*^2 := \frac{1}{MN} \sum_{m=1}^M \sum_{i=1}^N \|\nabla f_i^m(\mathbf{x}^*) - \nabla F^m(\mathbf{x}^*)\|^2, \quad (\text{B.10})$$

for the global minimum $\mathbf{x}^*$, and use it in their strong convexity and convexity bounds. Khaled et al. (2020) define

$$\sigma_{\text{opt}}^2 := \frac{1}{MN} \sum_{m=1}^M \sum_{i=1}^N \|\nabla f_i^m(\mathbf{x}^*)\|^2, \quad (\text{B.11})$$

which is used in their bound on strongly convex functions in homogeneous cases. Note that for homogeneous cases, $\nabla F^m(\mathbf{x}^*) = \nabla F(\mathbf{x}^*) = \mathbf{0}$, so $\bar{\sigma}_*^2 = \sigma_{\text{opt}}^2$.

We note that in contrast to our discussion on inter-machine deviation (Assumption 5.3), the intra-machine variance assumptions in the existing literature are weaker than our Assumption 5.2. However, we utilize our stronger assumption to prove our *high-probability* upper bounds, which is a departure from *in-expectation* bounds in the literature.

Existing upper bounds on local SGD. In the discussion after Theorem 5.7, we mentioned some recent upper bounds on (with-replacement) local SGD. We make more detailed comparisons here. For the reader's convenience, we restate our theorem on local RR below.

Theorem 5.7 (Upper bound for local RR). *Suppose that local RR has parameters satisfying Assumption 5.1. For any $F \in \mathcal{F}_{obj}(L, \mu, \nu, \tau, \rho)$, consider running the algorithm using step-size $\eta = \frac{\log(MNK^2)}{\mu NK}$ for epochs $K \geq 7\rho\kappa \log(MNK^2)$. Then, with probability at least $1 - \delta$,*

$$F(\mathbf{y}_{K, \frac{N}{B}}) - F^* \leq \frac{F(\mathbf{y}_0) - F^*}{MNK^2} + \tilde{\mathcal{O}} \left(\frac{L^2}{\mu^3} \left(\frac{\nu^2}{MNK^2} + \frac{\nu^2 B}{N^2 K^2} + \frac{\tau^2 B^2}{N^2 K^2} \right) \right). \quad (5.5)$$

We start with last-iterate bounds in homogeneous cases. Theorem 3 and Corollary 3 of Khaled et al. (2020) consider local SGD on μ -strongly convex and L -smooth F . Khaled et al. (2020) allow variable synchronization intervals, where the maximum interval is upper-bounded by B . In this setting, Corollary 3 of Khaled et al. (2020) shows that local SGD after T total local update steps yield

$$\mathbb{E} [\|\bar{\mathbf{x}}_T - \mathbf{x}_*\|^2] = \tilde{\mathcal{O}} \left(\frac{\|\bar{\mathbf{x}}_0 - \mathbf{x}_*\|^2}{T^2} + \frac{\sigma_{\text{opt}}^2}{\mu^2 MT} + \frac{L\sigma_{\text{opt}}^2(B-1)}{\mu^3 T^2} \right), \quad (B.12)$$

where $\bar{\mathbf{x}}_i$ is the average over all M machines' i -th local iterates, and σ_{opt} is from (B.11). Noting that T corresponds to NK in local SGD and σ_{opt} corresponds to ν in Assumption 5.2, (B.12) is comparable to an upper bound¹

$$\mathbb{E} [F(\bar{\mathbf{x}}_T) - F^*] = \tilde{\mathcal{O}} \left(\frac{L\nu^2}{\mu^2 MNK} + \frac{L^2\nu^2 B}{\mu^3 N^2 K^2} \right). \quad (B.13)$$

Here, we do not compare $\frac{F(\mathbf{y}_0) - F^*}{MNK^2}$ and $\frac{L\|\bar{\mathbf{x}}_0 - \mathbf{x}_*\|^2}{T^2}$ because in Theorem 5.7, the term $\frac{F(\mathbf{y}_0) - F^*}{MNK^2}$ can be made arbitrarily "small" (e.g., $\frac{F(\mathbf{y}_0) - F^*}{(MNK)^l}$ for any $l \in \mathbb{N}$) by changing the log factor in η .

A similar homogeneous, strongly convex, and smooth setting is considered in Spiridonoff et al. (2020), under a intra-machine variance assumption defined in (B.9). Theorem 1 of Spiridonoff et al. (2020) proves general theorem statement for arbitrary synchronization intervals. If we specialize to constant interval B , Corollary 1 of Spiridonoff et al. (2020) gives

$$\mathbb{E} [F(\bar{\mathbf{x}}_T) - F^*] = \tilde{\mathcal{O}} \left(\frac{\beta^2(F(\bar{\mathbf{x}}_0) - F^*)}{T^2} + \frac{L\sigma^2}{\mu^2 MT} + \frac{L^2\sigma^2(B-1)}{\mu^3 T^2} \right), \quad (B.14)$$

where $\beta \geq 2\kappa^2$ is a constant defined to choose algorithm parameters such as the step-size. Noting again that T corresponds to NK and σ in (B.9) is comparable to ν in Assumption 5.2,

¹Note that an additional factor L is due to conversion from squared distance to function value.

(B.14) also translates to (B.13).

The next last-iterate bound we compare against is Qu et al. (2020). Theorem 1 of Qu et al. (2020) uses bounded intra-machine variance assumption (B.7) and bounded gradient assumption (B.3). Specializing Theorem 1 of Qu et al. (2020) to full device participation and uniform weight ($p_1 = \dots = p_N = \frac{1}{N}$) case, their bound reads

$$\mathbb{E}[F(\bar{\mathbf{x}}_T) - F^*] = \tilde{O}\left(\frac{L\sigma^2}{\mu^2 MT} + \frac{L^2 G^2 B^2}{\mu^3 T^2}\right). \quad (\text{B.15})$$

Recalling that T corresponds to NK , σ to ν in Assumption 5.2, and G to the heterogeneity constant τ in Assumption 5.3 and also ν (due to (B.8)), (B.15) translates to

$$\mathbb{E}[F(\bar{\mathbf{x}}_T) - F^*] = \tilde{O}\left(\frac{L\nu^2}{\mu^2 MNK} + \frac{L^2(\nu^2 + \tau^2)B^2}{\mu^3 N^2 K^2}\right). \quad (\text{B.16})$$

Comparing the local SGD last-iterate bounds (B.13) and (B.16) against our local RR bound (5.5) in Theorem 5.7, we can see that the last iterate of local RR satisfies a smaller upper bound as soon as $K \geq \kappa$. Admittedly, this is not a fully rigorous comparison given the differences in assumptions and types of bounds; nevertheless, we believe that the comparison at least provides some degree of evidence for faster convergence of local RR than local SGD. It is also interesting to see that the “error from local updates” terms match in with- and without-replacement bounds.

Next, we review existing average-iterate bounds mentioned in the main text, which are better than the last-iterate bounds. Koloskova et al. (2020) present a unifying framework for analyzing distributed optimization algorithms over networks, which can specialize to local SGD. For μ -strongly convex and L -smooth F , Theorem 2 of Koloskova et al. (2020) shows that

$$\mathbb{E}[F(\hat{\mathbf{x}}) - F^*] = \tilde{O}\left(LB \|\mathbf{x}_0 - \mathbf{x}^*\|^2 \exp\left(-\frac{\mu T}{LB}\right) + \frac{\bar{\sigma}_*^2}{\mu MT} + \frac{L\bar{\sigma}_*^2 B}{\mu^2 T^2} + \frac{L\bar{\zeta}_*^2 B^2}{\mu^2 T^2}\right), \quad (\text{B.17})$$

where $\hat{\mathbf{x}}$ is some weighted average of iterates and $\bar{\zeta}_*^2$ and $\bar{\sigma}_*^2$ are defined in (B.5) and (B.10), respectively. Noting that T corresponds to NK , $\bar{\zeta}_*$ to τ in Assumption 5.3, and $\bar{\sigma}_*$ to ν in Assumption 5.2 (although our assumptions are stronger), we can see that the bound (B.17) for large enough T can be translated into

$$\mathbb{E}[F(\hat{\mathbf{x}}) - F^*] = \tilde{O}\left(\frac{\nu^2}{\mu MNK} + \frac{L\nu^2 B}{\mu^2 N^2 K^2} + \frac{L\tau^2 B^2}{\mu^2 N^2 K^2}\right). \quad (\text{B.18})$$

Notice that (B.18) is smaller than the last-iterate bounds (B.13) and (B.16) by a factor of κ .

Similarly, Theorem 3 of Woodworth et al. (2020b) proves that for μ -strongly convex and L -smooth F ,

$$\mathbb{E}[F(\tilde{\mathbf{x}}) - F^*] = \tilde{\mathcal{O}} \left(\frac{L^2 \|\mathbf{x}_0 - \mathbf{x}^*\|^2}{LT + \mu T^2} + \frac{\bar{\sigma}_*^2}{\mu MT} + \frac{L\sigma^2 B}{\mu^2 T^2} + \frac{L\bar{\zeta}^2 B^2}{\mu^2 T^2} \right), \quad (\text{B.19})$$

for some weighted average of iterates $\tilde{\mathbf{x}}$. Here, $\bar{\sigma}_*^2$, σ^2 , and $\bar{\zeta}^2$ are as defined in (B.10), (B.7), and (B.4), respectively. Substituting ν to its comparable constants $\bar{\sigma}_*$ and σ , and τ to $\bar{\zeta}$, we can similarly check that (B.19) can be converted to (B.18).

Comparison to Mishchenko et al. (2021). In Mishchenko et al. (2021), the authors study a proximal algorithm referred to as Proximal Random Reshuffling (ProxRR), and obtain a distributed optimization algorithm called FedRR as a special case. If we set $R \equiv 0$ in FedRR, the algorithm then is equal to local RR with $B = N$, i.e., the one that synchronizes only after one entire epoch.

Assuming that all component functions $f_i^m(\mathbf{x})$ are μ -strongly convex² and L -smooth, and objective-wise homogeneity $F^1 = \dots = F^m = F$, the authors obtain Theorem 8 (Mishchenko et al., 2021), which states that

$$\mathbb{E}[\|\mathbf{y}_K - \mathbf{x}^*\|^2] \leq (1 - \eta\mu)^{NK} \|\mathbf{y}_0 - \mathbf{x}^*\|^2 + \frac{\eta^2 LMN\sigma_{\text{opt}}^2}{\mu M}, \quad (\text{B.20})$$

where $\mathbf{y}_0$ and $\mathbf{y}_K$ are the initialization and last iterate of the algorithm, and σ_{opt}^2 was defined above in (B.11). The term $MN\sigma_{\text{opt}}^2$ in (B.20) corresponds to the term “ $N\sigma_*^2$ ” as per the notation in Mishchenko et al. (2021). If we apply our Assumption 5.3 to bound σ_{opt}^2 , we get $\sigma_{\text{opt}}^2 \leq \nu^2$, which reduces the last term in (B.20) to $\frac{\eta^2 L\nu^2 N}{\mu}$. If we substitute $\eta = \frac{\log(MNK^2)}{\mu NK}$ to the bound (B.20), we get

$$\mathbb{E}[\|\mathbf{y}_K - \mathbf{x}^*\|^2] \leq \frac{\|\mathbf{y}_0 - \mathbf{x}^*\|^2}{MNK^2} + \tilde{\mathcal{O}} \left(\frac{L\nu^2}{\mu^3 NK^2} \right), \quad (\text{B.21})$$

which translates to the same convergence rate on $F(\mathbf{y}_K) - F^*$ as single-machine RR. For the heterogeneous setting, applying Lemma 3 of Mishchenko et al. (2021) to Theorem 2, we can obtain

$$\begin{aligned} \mathbb{E}[\|\mathbf{y}_K - \mathbf{x}^*\|^2] &\leq (1 - \eta\mu)^{NK} \|\mathbf{y}_0 - \mathbf{x}^*\|^2 \\ &\quad + \frac{2\eta^2 L}{\mu M} \sum_{m=1}^M \left(N^2 \|\nabla F^m(\mathbf{x}^*)\|^2 + \frac{1}{4} \sum_{i=1}^N \|\nabla f_i^m(\mathbf{x}^*) - \nabla F^m(\mathbf{x}^*)\|^2 \right) \end{aligned}$$

²This is in fact quite strong compared to this paper, because we only assume F to be PL, not F^m nor f_i^m .

$$= (1 - \eta\mu)^{NK} \|\mathbf{y}_0 - \mathbf{x}^*\|^2 + \frac{2\eta^2 LN^2}{\mu M} \sum_{m=1}^M \|\nabla F^m(\mathbf{x}^*)\|^2 + \frac{\eta^2 L \bar{\sigma}_*^2 N}{2\mu}, \quad (\text{B.22})$$

where $\bar{\sigma}_*^2$ was defined in (B.10). Note that in Lemma 3 (Mishchenko et al., 2021), the function “ F_m ” in the authors’ notation is equal to NF^m in our notation. Recall that the (G, B) -BGD assumption (B.2) is “comparable” to Assumption 5.3 (B.1). If we apply (B.2) to the bound (B.22), we get $\frac{1}{M} \sum_{m=1}^M \|\nabla F^m(\mathbf{x}^*)\|^2 \leq G^2$. Similarly, if we apply Assumption 5.3, we get $\bar{\sigma}_*^2 \leq \nu^2$. Substituting these upper bounds and $\eta = \frac{\log(MNK^2)}{\mu NK}$ to (B.22) gives

$$\mathbb{E} [\|\mathbf{y}_K - \mathbf{x}^*\|^2] \leq \frac{\|\mathbf{y}_0 - \mathbf{x}^*\|^2}{MNK^2} + \tilde{\mathcal{O}} \left(\frac{LG^2}{\mu^3 K^2} + \frac{L\nu^2}{\mu^3 NK^2} \right), \quad (\text{B.23})$$

and after translating this into a bound on function value, we get an upper bound $\tilde{\mathcal{O}}(\frac{L^2\nu^2}{\mu^3 NK^2} + \frac{L^2G^2}{\mu^3 K^2})$ which in fact matches our upper bound (5.5) in Theorem 5.7 when we set $B = N$.

The two upper bounds obtained for homogeneous (B.21) and heterogeneous (B.23) settings indicate that, at least under assumptions on intra- and inter-machine deviation such as ours, the claimed advantage that “the convergence bound improves with the number of devices involved” (Mishchenko et al., 2021) is not achievable. As our lower bound shows, one needs to choose B smaller than N in order to get the most out of parallelism. That being said, since Mishchenko et al. (2021) is free of uniform intra- and inter-machine deviation assumptions, there may still exist certain scenarios where multiple machines can speed up performance even with $B = N$.

B.2 More detailed illustration of synchronized shuffling

In this section, we provide a more detailed explanation on synchronized shuffling that we introduced in Section 5.5. For the illustration, let us consider the component-wise homogeneous case. Component-wise homogeneity means that all the machines have the same set of components: $f_i^1 = f_i^2 = \dots = f_i^M =: f_i$ for $i \in [N]$. Hence, we have $F = F^m$ for all $m \in [M]$ and our goal is to minimize $F = \frac{1}{N} \sum_{i=1}^N f_i$.

In the proof of Theorem 1 (presented in Appendix B.4.2; see Appendix B.4.1 for a sketch), we add the component gradients over an epoch and then use the following key identity: for any permutation σ ,

$$\sum_{i=1}^N \nabla f_{\sigma(i)} = \sum_{i=1}^N \nabla f_i = N \nabla F. \quad (\text{B.24})$$

We use this identity (B.24) to represent the per-epoch progress as “one big GD step plus noise.” For the rest of the proof we bound the “noise” term, and the key to bounding it

is to upper bound the norm of summations of the following form, for $i \in [N/B - 1]$ (see (B.30) and (B.32)):

$$\frac{1}{M} \sum_{m=1}^M \sum_{j=1}^{iB} \nabla f_{\sigma_k^m(j)}(\mathbf{x}_{k,0}). \quad (\text{B.25})$$

To bound the norm, we decompose it into two terms using the triangle inequality

$$\begin{aligned} & \left\| \frac{1}{M} \sum_{m=1}^M \sum_{j=1}^{iB} \nabla f_{\sigma_k^m(j)}(\mathbf{x}_{k,0}) \right\| \\ & \leq \left\| \frac{1}{M} \sum_{m=1}^M \sum_{j=1}^{iB} \nabla f_{\sigma_k^m(j)}(\mathbf{x}_{k,0}) - iB \nabla F(\mathbf{x}_{k,0}) \right\| + iB \|\nabla F(\mathbf{x}_{k,0})\|, \end{aligned}$$

and we use concentration bounds (B.33) on the first term of the RHS, which gives a high-probability upper bound on the RHS of the form $\tilde{O}\left(\nu\sqrt{\frac{iB}{M}}\right) + iB \|\nabla F\|$ (B.34).

The key to proving fast convergence is to make the upper bound above as small as possible. To make the bound (B.34) even smaller, we wish to be able to apply the identity (B.24) to the summation (B.25). However, under the standard way of choosing permutations σ_k^m independently over machines, one cannot apply the identity because we do not sum over all $j = 1, \dots, N$. This limitation motivates our proposed technique *synchronized shuffling*, a manipulation on the choices of σ_k^m that lets us prove even faster convergence.

Recall the definition of synchronized shuffling. At the beginning of the k -th epoch, the server samples $\sigma \sim \text{Unif}(\mathcal{S}_N)$ and $\pi \sim \text{Unif}(\mathcal{S}_M)$, and broadcasts them to the machines. Then, local machines choose their permutations σ_k^m to be *shifted* versions of σ : $\sigma_k^m(i) := \sigma\left(\left(i + \frac{N}{M}\pi(m)\right) \bmod N\right)$.

Now set $N = 6$ and $M = 3$. Assume for simplicity that the permutation $\pi \in \mathcal{S}_3$ of machines satisfies $\pi(m) = m$ for $m \in [3]$.³ Suppose the server samples $\sigma = (\sigma(1), \sigma(2), \sigma(3), \sigma(4), \sigma(5), \sigma(6))$ and broadcasts it. Under synchronized shuffling, the local machines choose

$$\begin{aligned} \sigma_k^1 &= (\sigma(3), \sigma(4), \sigma(5), \sigma(6), \sigma(1), \sigma(2)), \\ \sigma_k^2 &= (\sigma(5), \sigma(6), \sigma(1), \sigma(2), \sigma(3), \sigma(4)), \\ \sigma_k^3 &= (\sigma(1), \sigma(2), \sigma(3), \sigma(4), \sigma(5), \sigma(6)). \end{aligned}$$

One can see that each permutation is a shifted version of σ , with an offset that is a multiple of $\frac{N}{M} = 2$.

Now consider adding $\nabla f_{\sigma_k^m(i)}$ over $m = 1, 2, 3$ and $j = 1, 2$ (in fact, any two consecutive

³In fact, permuting the machines by π is not required in the component-wise homogeneous setting (i.e., when $\lambda = 0$ in Assumption 5.4).

j 's will do). By synchronized shuffling, we get a summation over all $N = 6$ component functions, which by (B.24) gives us the full gradient:

$$\sum_{m=1}^3 \sum_{j=1}^2 \nabla f_{\sigma_k^m(j)} = \sum_{i=1}^6 \nabla f_i = 6 \nabla F.$$

The point here is that the permutation identity (B.24) can be applied to the summations (B.25) to further reduce their norm bounds. This is in contrast to sampling independent σ_k^m 's where one cannot apply (B.24). For this reason, synchronized shuffling significantly reduces the noise that comes from without-replacement sampling, thus resulting in faster convergence rates in (near-)homogeneous cases.

B.3 Experimental results

In this section, we present some simple numerical experiments that support our theoretical analysis. We evaluate the performance of the algorithms considered in this paper on the “hard instance” constructed in our lower bounds (Theorems 5.8 and 5.10).

Our hard instance $F \in \mathcal{F}_{\text{cmp}}(L, \mu, \nu, 0)$ is a function in the component-wise homogeneous setting, where all the machines have the same set of local component functions: $f_i^1 = f_i^2 = \dots = f_i^M =: f_i$. In the proofs of Theorems 5.8 and 5.10, we construct the global objective $F(x) = \frac{1}{N} \sum_{i=1}^N f_i(x)$ as the following:

$$f_i(x) := (L1_{x \leq 0} + \mu 1_{x > 0}) \frac{x^2}{2} + z_i \nu x, \quad z_i = \begin{cases} +1 & \text{if } 1 \leq i \leq N/2, \\ -1 & \text{if } N/2 < i \leq N. \end{cases}$$

With this set of component functions, the global objective $F(x) = (L1_{x \leq 0} + \mu 1_{x > 0}) \frac{x^2}{2}$ is μ -strongly convex and L -smooth with a unique global minimizer at $x = 0$.

We compare the performance of the algorithms on this problem instance, with $L = 100$, $\mu = 1$, $\nu = 1$, $N = 768$, and $M = 16$, while varying the choice of $B \in \{1, 4, 16, 64, 256\}$ and $K \in \{1, 3, 5, 7, 10, 30, 50, 70, 100, 300, 500, 700, 1000\}$. For each value of B and K , we run the algorithms for K epochs (KN/B communication rounds for with-replacement algorithms) starting at $x_0 = 0$ and return the values of F evaluated at the last iterates. Note that the algorithms are not deterministic, because the sampling/shuffling schemes are random. In order to account for randomness, for each combination of (algorithm, B , K) we execute 20 independent runs of the algorithm and plot the mean, first quartile, and third quartile of the final objective values.

In the subsequent subsections, we compare the following seven algorithms with constant step-sizes.

- Minibatch RR, $\eta = \frac{B \log(MNK^2)}{\mu NK}$;
- Local RR, $\eta = \frac{\log(MNK^2)}{\mu NK}$;
- Minibatch RR with SYNC SHUF, $\eta = \frac{B \log(MNK^2)}{\mu NK}$;
- Local RR with SYNC SHUF, $\eta = \frac{\log(MNK^2)}{\mu NK}$;
- Single-machine RR with minibatch size B , $\eta = \frac{\log(NK^2)}{\mu NK}$;
- With-replacement minibatch SGD, $\eta = \frac{B \log(MNK^2)}{\mu NK}$;
- With-replacement local SGD, $\eta = \frac{\log(MNK^2)}{\mu NK}$.

B.3.1 SYNC SHUF improves convergence of minibatch/local RR

In Figure B.1, we compare minibatch RR and local RR, with and without SYNC SHUF. Each plot in Figure B.1 shows how the methods' performance changes with K , for a fixed value B . Each point on the curve is the mean of the final objective function values over 20 independent runs of the corresponding algorithm with the specific B and K , and its error bar indicates the first and third quartiles.

Recall from our Theorems 5.6, 5.7, 5.13, and 5.14 that the four methods satisfy the following convergence bounds, in homogeneous settings (i.e., $\tau = \lambda = 0$):

- Minibatch RR: $\tilde{O}\left(\frac{L^2}{\mu^3} \frac{\nu^2}{MNK^2}\right)$,
- Local RR: $\tilde{O}\left(\frac{L^2}{\mu^3} \left(\frac{\nu^2}{MNK^2} + \frac{\nu^2 B}{N^2 K^2}\right)\right)$,
- Minibatch RR with SYNC SHUF: $\tilde{O}\left(\frac{L^2}{\mu^3} \frac{\nu^2}{M^2 N K^2}\right)$,
- Local RR with SYNC SHUF: $\tilde{O}\left(\frac{L^2}{\mu^3} \left(\frac{\nu^2}{M^2 N K^2} + \frac{\nu^2 B}{N^2 K^2}\right)\right)$.

In fact, if $B = 1$, local RR is identical to minibatch RR. Figure B.1(a) confirms that this is indeed true, and also that the versions with SYNC SHUF outperforms the ones without SYNC SHUF. This corroborates the additional M factor speedup in our bounds. In Figure B.1(b) and B.1(c), we can see that as B increases, the performance of local RR with SYNC SHUF degrades and becomes closer to local RR without SYNC SHUF. This shows that the $\tilde{O}\left(\frac{L^2}{\mu^3} \frac{\nu^2 B}{N^2 K^2}\right)$ term starts to dominate. Also, as we increase B further, in Figure B.1(c) and B.1(d) we see that local RR (without SYNC SHUF) also starts to degrade and its gap between

minibatch RR becomes larger. Again, this means that the $\tilde{O}\left(\frac{L^2}{\mu^3} \frac{\nu^2 B}{N^2 K^2}\right)$ term becomes the dominant factor in the local RR bound. The performance of minibatch RR, with and without SYNC SHUF, stays relatively independent of B . One thing to note is that for large values of B , the small-epoch behavior of minibatch RR looks rather unstable. The choice of step-size $\eta = \frac{B \log(MNK^2)}{\mu NK}$ seems to cause overshooting when B is large and K is small. We note that this does not contradict our convergence analysis because our theorems only characterize the large-epoch behavior (K above certain thresholds) of the algorithms. Perhaps in the small-epoch regime, our choice of η is not necessarily optimal and a smaller η is needed to prevent overshooting.

B.3.2 Local RR becomes closer to single-machine RR as $B \rightarrow N$

Our next set of plots presented in Figure B.2 provides a comparison of minibatch RR, local RR, and single-machine RR (i.e., minibatch RR with $M = 1$). In Theorems 5.7 and 5.10, we showed that when $B = \Theta(N)$, then the convergence of local RR becomes just as fast as the single-machine RR. Indeed, we can observe from Figure B.2 that this is really the case. As B increases, the curve of local RR moves closer and closer to that of single-machine RR, especially in the large-epoch regime.

B.3.3 With- vs. without-replacement sampling

Lastly, in Figure B.3 we compare the with-replacement and without-replacement versions of minibatch/local SGD. In all plots, we can see that the without-replacement versions outperform with-replacement ones, at least for our problem instance. It is also intriguing to note that the two versions perform very similarly in the small-epoch regime (for K up to ~ 10), but without-replacement starts to outperform for larger K 's. This observation supports our theoretical prediction from Theorem 5.8 that in the small-epoch regime, minibatch RR can at best perform as fast as minibatch SGD.

B.4 Proofs of upper bounds

In this section, we provide proofs of our upper bounds stated in Sections 5.3 and 5.5. We start by describing a high-level proof outline that we use for all the proofs presented in this section (Appendix B.4.1). In the subsequent subsections, we prove Theorems 5.6, 5.7, 5.13, and 5.14, in the order they appeared in the main text. The next subsection (Appendix B.4.6) states and proves a key lemma that gives concentration bounds for the mean of multiple without-replacement sums of vectors. This general-purpose lemma can

be of independent interest and it can prove useful in various other settings. Lastly, in Appendix B.4.7 we discuss how we can modify the theorem statements to remove the requirement in Assumptions 5.2, 5.3, and 5.4 that they must hold for the entire $\mathbb{R}^d$.

Notation. Throughout this section, we use the product notation $\prod$ in a slightly unconventional manner. For indices $i \leq j$ and square matrices $\mathbf{A}_i, \mathbf{A}_{i+1}, \dots, \mathbf{A}_{j-1}, \mathbf{A}_j$, we use $\prod_{l=i}^j \mathbf{A}_l$ to denote the matrix product $\mathbf{A}_j \mathbf{A}_{j-1} \cdots \mathbf{A}_{i+1} \mathbf{A}_i$. If $i > j$, then $\prod_{l=i}^j \mathbf{A}_l = \mathbf{I}$.

B.4.1 Proof outline

The proofs of upper bounds follow a common structure, consisting of the following three steps:

1. writing one epoch as one step of GD plus noise;
2. getting a high-probability upper bound on the noise term using concentration inequalities;
3. obtaining the convergence rate using the bounds on the noise term.

We first unroll the update equations over an epoch, and write an epoch of the algorithms as one step of gradient descent plus noise:⁴

$$\mathbf{x}_{k+1,0} = \mathbf{x}_{k,0} - \eta N \nabla F(\mathbf{x}_{k,0}) + \eta^2 \mathbf{r}_k.$$

Substituting the above to the definition of L -smoothness of F and arranging terms, we obtain

$$\begin{aligned} & F(\mathbf{x}_{k+1,0}) - F(\mathbf{x}_{k,0}) \\ & \leq \langle \nabla F(\mathbf{x}_{k,0}), \mathbf{x}_{k+1,0} - \mathbf{x}_{k,0} \rangle + \frac{L}{2} \|\mathbf{x}_{k+1,0} - \mathbf{x}_{k,0}\|^2 \\ & \leq -\eta N \|\nabla F(\mathbf{x}_{k,0})\|^2 + \eta^2 \|\nabla F(\mathbf{x}_{k,0})\| \|\mathbf{r}_k\| + \frac{\eta^2 L}{2} \|N \nabla F(\mathbf{x}_{k,0}) + \eta \mathbf{r}_k\|^2 \\ & \leq (-\eta N + \eta^2 L N^2) \|\nabla F(\mathbf{x}_{k,0})\|^2 + \eta^2 \|\nabla F(\mathbf{x}_{k,0})\| \|\mathbf{r}_k\| + \eta^4 L \|\mathbf{r}_k\|^2. \end{aligned} \tag{B.26}$$

The next step is to get high-probability upper bounds on the term $\|\mathbf{r}_k\|$. This is done by applying our concentration inequality lemma (Lemma B.1) to partial without-replacement sums of component gradients. As a result, we will get upper bounds on $\|\mathbf{r}_k\|$ and $\|\mathbf{r}_k\|^2$, for $k = 1, \dots, K$, which hold with probability at least $1 - \delta$.

⁴In case of local RR, we replace $\mathbf{x}_{k,0}$ with $\mathbf{y}_{k,0}$.

In the last part, we substitute the high-probability bounds to (B.26) and invoke the definition of PL functions to get a per-epoch progress bound. We then unroll the per-epoch inequality for all epochs $k = 1, \dots, K$. Arranging the terms in the resulting inequality gives our desired convergence bound that holds with probability at least $1 - \delta$.

B.4.2 Proof of upper bound for minibatch RR (Theorem 5.6)

One epoch as one step of GD plus noise. To simplify the notation throughout the proof, we will prove the same convergence rate for a rescaled update rule and step-size:

$$\mathbf{x}_{k,i} := \mathbf{x}_{k,i-1} - \frac{\eta}{M} \sum_{m=1}^M \sum_{j=(i-1)B+1}^{iB} \nabla f_{\sigma_k^m(j)}^m(\mathbf{x}_{k,i-1}) \quad (\text{B.27})$$

for $i \in [N/B]$ and $k \in [K]$, and $\eta = \frac{\log(MNK^2)}{\mu NK}$. Note that the gradient term is scaled up by B and the step-size is scaled down by B . We will prove the convergence rate for this equivalent algorithm.

We start the proof by unrolling the update equations over an epoch and expressing the progress as

$$\mathbf{x}_{k+1,0} = \mathbf{x}_{k,0} - \eta NF(\mathbf{x}_{k,0}) + \eta^2 \mathbf{r}_k,$$

i.e., one step of full gradient descent plus some noise.

To this end, we decompose the gradient $\nabla f_{\sigma_k^m(j)}^m(\mathbf{x}_{k,i-1})$ into signal $\nabla f_{\sigma_k^m(j)}^m(\mathbf{x}_{k,0})$ and noise:

$$\begin{aligned} \nabla f_{\sigma_k^m(j)}^m(\mathbf{x}_{k,i-1}) &= \nabla f_{\sigma_k^m(j)}^m(\mathbf{x}_{k,0}) + \nabla f_{\sigma_k^m(j)}^m(\mathbf{x}_{k,i-1}) - \nabla f_{\sigma_k^m(j)}^m(\mathbf{x}_{k,0}) \\ &= \nabla f_{\sigma_k^m(j)}^m(\mathbf{x}_{k,0}) + \left[\int_0^1 \nabla^2 f_{\sigma_k^m(j)}^m(\mathbf{x}_{k,0} + t(\mathbf{x}_{k,i-1} - \mathbf{x}_{k,0})) dt \right] (\mathbf{x}_{k,i-1} - \mathbf{x}_{k,0}), \end{aligned}$$

where $\nabla^2 f(\mathbf{x})$ denotes the Hessian of f at $\mathbf{x}$, whenever it exists. We remark that the integral exists, due to the following reason. Since we assumed that each $f_{\sigma_k^m(j)}^m$ is differentiable and smooth, its gradient $\nabla f_{\sigma_k^m(j)}^m$ is Lipschitz continuous, and hence absolutely continuous. This means that $\nabla f_{\sigma_k^m(j)}^m$ is differentiable almost everywhere (i.e., $\nabla^2 f_{\sigma_k^m(j)}^m(\mathbf{x})$ exists a.e.), and the fundamental theorem of calculus for Lebesgue integral holds; hence the integral exists.

To simplify notation, we define the following for all $i \in [N/B]$:

$$\mathbf{g}_i := \frac{1}{M} \sum_{m=1}^M \sum_{j=(i-1)B+1}^{iB} \nabla f_{\sigma_k^m(j)}^m(\mathbf{x}_{k,0}),$$

$$\mathbf{H}_i := \frac{1}{M} \sum_{m=1}^M \sum_{j=(i-1)B+1}^{iB} \int_0^1 \nabla^2 f_{\sigma_k^m(j)}^m(\mathbf{x}_{k,0} + t(\mathbf{x}_{k,i-1} - \mathbf{x}_{k,0})) dt,$$

so that we can write (B.27) as

$$\mathbf{x}_{k,i} = \mathbf{x}_{k,i-1} - \eta \mathbf{g}_i - \eta \mathbf{H}_i(\mathbf{x}_{k,i-1} - \mathbf{x}_{k,0}). \quad (\text{B.28})$$

From L -smoothness of f_i^m 's, it is straightforward to check that $\|\mathbf{H}_i\| \leq LB$. Unrolling (B.28) for $i = 1, \dots, N/B$, it turns out that we can write

$$\mathbf{x}_{k+1,0} = \mathbf{x}_{k,0} - \eta \sum_{i=1}^{N/B} \left(\prod_{j=N/B}^{i+1} (\mathbf{I} - \eta \mathbf{H}_j) \right) \mathbf{g}_i.$$

Due to summation by parts, the following identity holds:

$$\sum_{i=1}^{N/B} a_i b_i = a_{N/B} \sum_{j=1}^{N/B} b_j - \sum_{i=1}^{N/B-1} (a_{i+1} - a_i) \sum_{j=1}^i b_j.$$

We apply this to the last term, by substituting $a_i = \prod_{j=N/B}^{i+1} (\mathbf{I} - \eta \mathbf{H}_j)$ and $b_i = \mathbf{g}_i$:

$$\begin{aligned} & \eta \sum_{i=1}^{N/B} \left(\prod_{j=N/B}^{i+1} (\mathbf{I} - \eta \mathbf{H}_j) \right) \mathbf{g}_i \\ &= \eta \sum_{j=1}^{N/B} \mathbf{g}_j - \eta \sum_{i=1}^{N/B-1} \left(\prod_{t=N/B}^{i+2} (\mathbf{I} - \eta \mathbf{H}_t) - \prod_{t=N/B}^{i+1} (\mathbf{I} - \eta \mathbf{H}_t) \right) \sum_{j=1}^i \mathbf{g}_j \\ &= \eta N \nabla F(\mathbf{x}_{k,0}) - \underbrace{\eta^2 \sum_{i=1}^{N/B-1} \left(\prod_{t=N/B}^{i+2} (\mathbf{I} - \eta \mathbf{H}_t) \right) \mathbf{H}_{i+1} \sum_{j=1}^i \mathbf{g}_j}_{=: \mathbf{r}_k}. \end{aligned}$$

With the “noise” $\mathbf{r}_k$ defined as above, we can write $\mathbf{x}_{k+1,0} = \mathbf{x}_{k,0} - \eta N \nabla F(\mathbf{x}_{k,0}) + \eta^2 \mathbf{r}_k$, as desired. Next, it follows from L -smoothness of F that

$$\begin{aligned} & F(\mathbf{x}_{k+1,0}) - F(\mathbf{x}_{k,0}) \\ &\leq \langle \nabla F(\mathbf{x}_{k,0}), \mathbf{x}_{k+1,0} - \mathbf{x}_{k,0} \rangle + \frac{L}{2} \|\mathbf{x}_{k+1,0} - \mathbf{x}_{k,0}\|^2 \\ &\leq -\eta N \|\nabla F(\mathbf{x}_{k,0})\|^2 + \eta^2 \|\nabla F(\mathbf{x}_{k,0})\| \|\mathbf{r}_k\| + \frac{\eta^2 L}{2} \|N \nabla F(\mathbf{x}_{k,0}) + \eta \mathbf{r}_k\|^2 \\ &\leq (-\eta N + \eta^2 L N^2) \|\nabla F(\mathbf{x}_{k,0})\|^2 + \eta^2 \|\nabla F(\mathbf{x}_{k,0})\| \|\mathbf{r}_k\| + \eta^4 L \|\mathbf{r}_k\|^2, \end{aligned} \quad (\text{B.29})$$

where the last inequality used $\|\mathbf{a} + \mathbf{b}\|^2 \leq 2\|\mathbf{a}\|^2 + 2\|\mathbf{b}\|^2$.

Bounding noise term using concentration. It is left to bound $\|\mathbf{r}_k\|$. We have

$$\begin{aligned}
\|\mathbf{r}_k\| &= \left\| \sum_{i=1}^{N/B-1} \left(\prod_{t=N/B}^{i+2} (\mathbf{I} - \eta \mathbf{H}_t) \right) \mathbf{H}_{i+1} \sum_{j=1}^i \mathbf{g}_j \right\| \\
&\leq \sum_{i=1}^{N/B-1} \left\| \left(\prod_{t=N/B}^{i+2} (\mathbf{I} - \eta \mathbf{H}_t) \right) \mathbf{H}_{i+1} \sum_{j=1}^i \mathbf{g}_j \right\| \\
&\leq LB(1 + \eta LB)^{N/B} \sum_{i=1}^{N/B-1} \left\| \sum_{j=1}^i \mathbf{g}_j \right\|, \tag{B.30}
\end{aligned}$$

where the last step used $\|\mathbf{H}_i\| \leq LB$ for $i \in [N/B]$. Recall from the theorem statement that $K \geq 6\kappa \log(MNK^2)$ and $\eta = \frac{\log(MNK^2)}{\mu NK}$. This means that

$$(1 + \eta LB)^{N/B} = \left(1 + \frac{\kappa B \log(MNK^2)}{NK} \right)^{N/B} \leq \left(1 + \frac{B}{6N} \right)^{N/B} \leq e^{1/6}. \tag{B.31}$$

Now, we use Lemma B.1 to bound the norm of

$$\sum_{j=1}^i \mathbf{g}_j = \frac{1}{M} \sum_{m=1}^M \sum_{j=1}^{iB} \nabla f_{\sigma_k^m(j)}^m(\mathbf{x}_{k,0}). \tag{B.32}$$

Note that for any epoch k , the permutations $\sigma_k^1, \dots, \sigma_k^M$ are independent of the first iterate $\mathbf{x}_{k,0}$ of the epoch, and hence independent of all $\nabla f_i^m(\mathbf{x}_{k,0})$. Therefore, we can apply Lemma B.1 to the partial sum (B.32), with $\mathbf{v}_i^m \leftarrow \nabla f_i^m(\mathbf{x}_{k,0})$, $n \leftarrow iB$, and $\delta \leftarrow \frac{B\delta}{NK}$. By Lemma B.1, with probability at least $1 - \frac{B\delta}{NK}$, we have

$$\left\| \frac{1}{iBM} \sum_{m=1}^M \sum_{j=1}^{iB} \nabla f_{\sigma_k^m(j)}^m(\mathbf{x}_{k,0}) - \nabla F(\mathbf{x}_{k,0}) \right\| \leq \nu \sqrt{\frac{8 \log \frac{2NK}{B\delta}}{iBM}}. \tag{B.33}$$

Using this concentration bound, with probability at least $1 - \frac{B\delta}{NK}$ we have

$$\left\| \sum_{j=1}^i \mathbf{g}_j \right\| \leq iB\nu \sqrt{\frac{8 \log \frac{2NK}{B\delta}}{iBM}} + iB \|\nabla F(\mathbf{x}_{k,0})\| = \nu \sqrt{\frac{8iB \log \frac{2NK}{B\delta}}{M}} + iB \|\nabla F(\mathbf{x}_{k,0})\|. \tag{B.34}$$

We can now substitute (B.31) and (B.34) to (B.30) to get

$$\|\mathbf{r}_k\| \leq e^{1/6} LB \sum_{i=1}^{N/B-1} \left(\nu \sqrt{\frac{8iB \log \frac{2NK}{B\delta}}{M}} + iB \|\nabla F(\mathbf{x}_{k,0})\| \right)$$

$$\begin{aligned}
&\leq \frac{e^{1/6}\sqrt{8}L\nu B^{3/2}}{M^{1/2}} \int_1^{N/B} \sqrt{t} dt \sqrt{\log \frac{2NK}{B\delta}} + e^{1/6}LB^2 \|\nabla F(\mathbf{x}_{k,0})\| \sum_{i=1}^{N/B-1} i \\
&\leq \frac{5L\nu(N^{3/2} - B^{3/2})}{2M^{1/2}} \sqrt{\log \frac{2NK}{B\delta}} + LN(N-B) \|\nabla F(\mathbf{x}_{k,0})\|, \tag{B.35}
\end{aligned}$$

which holds with probability at least $1 - \frac{\delta}{K}$, due to the union bound over $i = 1, \dots, N/B - 1$. The bound (B.35) holds for all $k \in [K]$ with probability $1 - \delta$ if we apply the union bound over $k = 1, \dots, K$. Next, by $(a + b)^2 \leq 2a^2 + 2b^2$, we have

$$\|\mathbf{r}_k\|^2 \leq \frac{25L^2\nu^2(N^{3/2} - B^{3/2})^2}{2M} \log \frac{2NK}{B\delta} + 2L^2N^2(N-B)^2 \|\nabla F(\mathbf{x}_{k,0})\|^2, \tag{B.36}$$

which also holds for all $k \in [K]$ with probability at least $1 - \delta$.

Getting a high-probability convergence rate. Given our high-probability bounds (B.35) and (B.36), we can substitute them to (B.29) and get

$$\begin{aligned}
&F(\mathbf{x}_{k+1,0}) - F(\mathbf{x}_{k,0}) \\
&\leq (-\eta N + \eta^2 L N^2) \|\nabla F(\mathbf{x}_{k,0})\|^2 + \eta^2 \|\nabla F(\mathbf{x}_{k,0})\| \|\mathbf{r}_k\| + \eta^4 L \|\mathbf{r}_k\|^2 \\
&\leq \left(-\eta N + \eta^2 L N^2 + \eta^2 L N(N-B) + 2\eta^4 L^3 N^2(N-B)^2 \right) \|\nabla F(\mathbf{x}_{k,0})\|^2 \\
&\quad + \frac{5\eta^2 L \nu(N^{3/2} - B^{3/2})}{2M^{1/2}} \sqrt{\log \frac{2NK}{B\delta}} \|\nabla F(\mathbf{x}_{k,0})\| \\
&\quad + \frac{25\eta^4 L^3 \nu^2(N^{3/2} - B^{3/2})^2}{2M} \log \frac{2NK}{B\delta}. \tag{B.37}
\end{aligned}$$

The second term in the RHS of (B.37) can be bounded using $ab \leq \frac{a^2}{2} + \frac{b^2}{2}$:

$$\begin{aligned}
&\frac{5\eta^2 L \nu(N^{3/2} - B^{3/2})}{2M^{1/2}} \sqrt{\log \frac{2NK}{B\delta}} \|\nabla F(\mathbf{x}_{k,0})\| \\
&= \left(\frac{\eta^{1/2} N^{1/2}}{2} \|\nabla F(\mathbf{x}_{k,0})\| \right) \left(\frac{5\eta^{3/2} L \nu(N^{3/2} - B^{3/2})}{M^{1/2} N^{1/2}} \sqrt{\log \frac{2NK}{B\delta}} \right) \\
&\leq \frac{\eta N}{8} \|\nabla F(\mathbf{x}_{k,0})\|^2 + \frac{25\eta^3 L^2 \nu^2(N^{3/2} - B^{3/2})^2}{2MN} \log \frac{2NK}{B\delta}.
\end{aligned}$$

Putting this inequality to (B.37) and noting $N - B \leq N$ gives

$$\begin{aligned}
F(\mathbf{x}_{k+1,0}) - F(\mathbf{x}_{k,0}) &\leq \left(-\frac{7}{8}\eta N + 2\eta^2 L N^2 + 2\eta^4 L^3 N^4 \right) \|\nabla F(\mathbf{x}_{k,0})\|^2 \\
&\quad + \frac{25\eta^3 L^2(1 + \eta L N)\nu^2(N^{3/2} - B^{3/2})^2}{2MN} \log \frac{2NK}{B\delta}. \tag{B.38}
\end{aligned}$$

Recall from $K \geq 6\kappa \log(MNK^2)$ and $\eta = \frac{\log(MNK^2)}{\mu NK}$ that $\eta LN \leq \frac{1}{6}$. Since the inequality $-\frac{7}{8}z + 2z^2 + 2z^4 \leq -\frac{1}{2}z$ holds on $z \in [0, \frac{1}{6}]$, we have

$$-\frac{7}{8}\eta N + 2\eta^2 L N^2 + 2\eta^4 L^3 N^4 \leq -\frac{1}{2}\eta N.$$

Applying this bound to (B.38) results in

$$F(\mathbf{x}_{k+1,0}) - F(\mathbf{x}_{k,0}) \leq -\frac{\eta N}{2} \|\nabla F(\mathbf{x}_{k,0})\|^2 + \frac{15\eta^3 L^2 \nu^2 (N^{3/2} - B^{3/2})^2}{MN} \log \frac{2NK}{B\delta}.$$

We now recall that F is μ -PL, so $\|\nabla F(\mathbf{x}_{k,0})\|^2 \geq 2\mu(F(\mathbf{x}_{k,0}) - F^*)$:

$$F(\mathbf{x}_{k+1,0}) - F^* \leq (1 - \eta\mu N)(F(\mathbf{x}_{k,0}) - F^*) + \frac{15\eta^3 L^2 \nu^2 (N^{3/2} - B^{3/2})^2}{MN} \log \frac{2NK}{B\delta}. \quad (\text{B.39})$$

Recall that (B.39) holds for all $k \in [K]$, with probability $1 - \delta$. Therefore, by unrolling the inequality,

$$\begin{aligned} F(\mathbf{x}_{K, \frac{N}{B}}) - F^* &\leq (1 - \eta\mu N)^K (F(\mathbf{x}_0) - F^*) \\ &\quad + \frac{15\eta^3 L^2 \nu^2 (N^{3/2} - B^{3/2})^2}{MN} \log \frac{2NK}{B\delta} \sum_{k=0}^{K-1} (1 - \eta\mu N)^k \\ &\leq (1 - \eta\mu N)^K (F(\mathbf{x}_0) - F^*) + \frac{15\eta^2 L^2 \nu^2 (N^{3/2} - B^{3/2})^2}{\mu MN^2} \log \frac{2NK}{B\delta}. \end{aligned} \quad (\text{B.40})$$

Lastly, substituting $\eta = \frac{\log(MNK^2)}{\mu NK}$ gives

$$\begin{aligned} F(\mathbf{x}_{K, \frac{N}{B}}) - F^* &\leq \frac{F(\mathbf{x}_0) - F^*}{MNK^2} + \frac{15L^2 \nu^2 (N^{3/2} - B^{3/2})^2 \log \frac{2NK}{B\delta} \log^2(MNK^2)}{\mu^3 MN^4 K^2} \\ &= \frac{F(\mathbf{x}_0) - F^*}{MNK^2} + \tilde{\mathcal{O}}\left(\frac{L^2 \nu^2}{\mu^3} \frac{1}{MNK^2}\right). \end{aligned} \quad (\text{B.41})$$

Getting an in-expectation bound from the high-probability bound. We conclude this subsection by briefly describing how we can obtain an in-expectation bound from the high-probability bound we just proved. Recall that the bound we proved above holds under the event E that all the concentration bounds used throughout the proof hold. The key to proving an in-expectation bound is to obtain an upper bound under its complement E^c , i.e., conditioned on the event that at least one of our concentration bounds does not hold. We do so by repeating the same proof *without* ever using the Hoeffding-Serfling bounds (Lemma B.1). Of course, this leads to a much looser bound, but we can choose δ to

be small enough so that the desired bound $\tilde{\mathcal{O}}\left(\frac{L^2\nu^2}{\mu^3}\frac{1}{MNK^2}\right)$ holds in expectation.

For the version without concentration bounds, the proof proceeds in the same way until it starts diverge at (B.33). Instead of applying concentration inequalities, we loosely bound the quantity as the following:

$$\begin{aligned} & \left\| \frac{1}{iBM} \sum_{m=1}^M \sum_{j=1}^{iB} \nabla f_{\sigma_k^m(j)}^m(\mathbf{x}_{k,0}) - \nabla F(\mathbf{x}_{k,0}) \right\| \\ &= \left\| \frac{1}{iBM} \sum_{m=1}^M \sum_{j=1}^{iB} \nabla f_{\sigma_k^m(j)}^m(\mathbf{x}_{k,0}) - \frac{1}{M} \sum_{m=1}^M \nabla F^m(\mathbf{x}_{k,0}) \right\| \\ &\leq \frac{1}{M} \sum_{m=1}^M \left\| \frac{1}{iB} \sum_{j=1}^{iB} \nabla f_{\sigma_k^m(j)}^m(\mathbf{x}_{k,0}) - \nabla F^m(\mathbf{x}_{k,0}) \right\| \leq \nu. \end{aligned}$$

With this bound, the RHS of the upper bound (B.34) on $\|\sum_{j=1}^i \mathbf{g}_j\|$ becomes $iB\nu + iB \|\nabla F(\mathbf{x}_{k,0})\|$. This results in the bounds on $\|\mathbf{r}_k\|$ and $\|\mathbf{r}_k\|^2$ (corresponding to (B.35) and (B.36)) that read

$$\begin{aligned} \|\mathbf{r}_k\| &\leq L\nu N(N-B) + LN(N-B) \|\nabla F(\mathbf{x}_{k,0})\|, \\ \|\mathbf{r}_k\|^2 &\leq 2L^2\nu^2 N^2(N-B)^2 + 2L^2 N^2(N-B)^2 \|\nabla F(\mathbf{x}_{k,0})\|^2. \end{aligned}$$

The rest is substituting the bounds above to (B.29), and going through the same steps to obtain the final bound. The resulting bound that corresponds to (B.40) is

$$F(\mathbf{x}_{K, \frac{N}{B}}) - F^* \leq (1 - \eta\mu N)^K (F(\mathbf{x}_0) - F^*) + \frac{7\eta^2 L^2 \nu^2 (N-B)^2}{3\mu},$$

which, by substituting $\eta = \frac{\log(MNK^2)}{\mu NK}$, yields

$$F(\mathbf{x}_{K, \frac{N}{B}}) - F^* \leq \frac{F(\mathbf{x}_0) - F^*}{MNK^2} + \tilde{\mathcal{O}}\left(\frac{L^2\nu^2}{\mu^3}\frac{1}{K^2}\right). \quad (\text{B.42})$$

To finish the proof of in-expectation bound, choose $\delta = \frac{1}{MN}$. Recall that the probabilistic event E occurs when all our concentration bounds hold. Conditioned on E , which occurs with probability at least $1 - \frac{1}{MN}$, the tighter bound (B.41) holds, with $\log \frac{1}{\delta}$ replaced by $\log(MN)$. The complement event E^c occurs with probability at most $\frac{1}{MN}$, under which the looser bound (B.42) is true. Thus, in expectation,

$$\begin{aligned} \mathbb{E} [F(\mathbf{x}_{K, \frac{N}{B}}) - F^*] &= \mathbb{P}(E) \mathbb{E} [F(\mathbf{x}_{K, \frac{N}{B}}) - F^* \mid E] + \mathbb{P}(E^c) \mathbb{E} [F(\mathbf{x}_{K, \frac{N}{B}}) - F^* \mid E^c] \\ &\leq \mathbb{E} [F(\mathbf{x}_{K, \frac{N}{B}}) - F^* \mid E] + \frac{1}{MN} \mathbb{E} [F(\mathbf{x}_{K, \frac{N}{B}}) - F^* \mid E^c] \end{aligned}$$

$$\leq \frac{3(F(\mathbf{x}_0) - F^*)}{2MNK^2} + \tilde{O}\left(\frac{L^2\nu^2}{\mu^3} \frac{1}{MNK^2}\right).$$

For the remaining high-probability upper bounds proved in the paper, we can similarly follow this process to obtain matching (up to log factors) in-expectation upper bounds.

B.4.3 Proof of upper bound for local RR (Theorem 5.7)

One epoch as one step of GD plus noise. The update rule of local RR can be written as the following. For $k \in [K]$, $i \in [N]$, and $m \in [M]$,

$$\mathbf{x}_{k,i}^m := \begin{cases} \mathbf{x}_{k,i-1}^m - \eta \nabla f_{\sigma_k^m(i)}^m(\mathbf{x}_{k,i-1}^m) & \text{if } B \text{ does not divide } i, \\ \frac{1}{M} \sum_{m=1}^M (\mathbf{x}_{k,i-1}^m - \eta \nabla f_{\sigma_k^m(i)}^m(\mathbf{x}_{k,i-1}^m)) =: \mathbf{y}_{k,i/B} & \text{if } B \text{ divides } i. \end{cases} \quad (\text{B.43})$$

Recall that $\mathbf{x}_{k,0}^m$'s are the initial points of an epoch and they all the same regardless of the machine m . We define $\mathbf{y}_{k,0} := \mathbf{x}_{k,0}^1$. For any i in the range of $(l-1)B + 1 \leq i \leq lB$ for some $l \in [M]$, we will use $\mathbf{y}_{k,l-1}$ as the “pivot” and decompose the gradients into the ones evaluated at $\mathbf{y}_{k,l-1}$ plus noise terms.

$$\begin{aligned} \nabla f_{\sigma_k^m(i)}^m(\mathbf{x}_{k,i-1}^m) &= \nabla f_{\sigma_k^m(i)}^m(\mathbf{y}_{k,l-1}) + \nabla f_{\sigma_k^m(i)}^m(\mathbf{x}_{k,i-1}^m) - \nabla f_{\sigma_k^m(i)}^m(\mathbf{y}_{k,l-1}) \\ &= \nabla f_{\sigma_k^m(i)}^m(\mathbf{y}_{k,l-1}) + \underbrace{\left[\int_0^1 \nabla^2 f_{\sigma_k^m(i)}^m(\mathbf{y}_{k,l-1} + t(\mathbf{x}_{k,i-1}^m - \mathbf{y}_{k,l-1})) dt \right]}_{=: \mathbf{H}_i^m} (\mathbf{x}_{k,i-1}^m - \mathbf{y}_{k,l-1}), \end{aligned}$$

where the integral $\mathbf{H}_i^m$ exists due to the reason discussed in Appendix B.4.2. Also note from L -smoothness of f_i^m 's that $\|\mathbf{H}_i^m\| \leq L$. Using the decomposition, one can unroll the updates (B.43) and write $\mathbf{y}_{k,l}$ in terms of $\mathbf{y}_{k,l-1}$ in the following way:

$$\mathbf{y}_{k,l} = \mathbf{y}_{k,l-1} - \frac{\eta}{M} \sum_{m=1}^M \sum_{i=(l-1)B+1}^{lB} \left(\prod_{j=lB}^{i+1} (\mathbf{I} - \eta \mathbf{H}_j^m) \right) \nabla f_{\sigma_k^m(i)}^m(\mathbf{y}_{k,l-1}), \quad (\text{B.44})$$

for $l = 1, \dots, N/B$. Next, we again decompose the gradient $\nabla f_{\sigma_k^m(i)}^m(\mathbf{y}_{k,l-1})$, this time using $\mathbf{y}_{k,0}$ as the pivot:

$$\begin{aligned} \nabla f_{\sigma_k^m(i)}^m(\mathbf{y}_{k,l-1}) &= \nabla f_{\sigma_k^m(i)}^m(\mathbf{y}_{k,0}) + \nabla f_{\sigma_k^m(i)}^m(\mathbf{y}_{k,l-1}) - \nabla f_{\sigma_k^m(i)}^m(\mathbf{y}_{k,0}) \\ &= \nabla f_{\sigma_k^m(i)}^m(\mathbf{y}_{k,0}) + \underbrace{\left[\int_0^1 \nabla^2 f_{\sigma_k^m(i)}^m(\mathbf{y}_{k,0} + t(\mathbf{y}_{k,l-1} - \mathbf{y}_{k,0})) dt \right]}_{=: \tilde{\mathbf{H}}_i^m} (\mathbf{y}_{k,l-1} - \mathbf{y}_{k,0}). \end{aligned}$$

This decomposition allows us to rewrite (B.44) in the following form

$$\mathbf{y}_{k,l} = \mathbf{y}_{k,l-1} - \eta \mathbf{t}_l - \eta \mathbf{S}_l(\mathbf{y}_{k,l-1} - \mathbf{y}_{k,0}), \quad (\text{B.45})$$

where

$$\mathbf{t}_l := \frac{1}{M} \sum_{m=1}^M \sum_{i=(l-1)B+1}^{lB} \left(\prod_{j=lB}^{i+1} (\mathbf{I} - \eta \mathbf{H}_j^m) \right) \nabla f_{\sigma_k^m(i)}^m(\mathbf{y}_{k,0}), \quad (\text{B.46})$$

$$\mathbf{S}_l := \frac{1}{M} \sum_{m=1}^M \sum_{i=(l-1)B+1}^{lB} \left(\prod_{j=lB}^{i+1} (\mathbf{I} - \eta \mathbf{H}_j^m) \right) \tilde{\mathbf{H}}_i^m. \quad (\text{B.47})$$

Unrolling (B.45) for $l = 1, \dots, N/B$ then gives the progress over an epoch:

$$\mathbf{y}_{k+1,0} = \mathbf{y}_{k,0} - \eta \sum_{l=1}^{N/B} \left(\prod_{j=N/B}^{l+1} (\mathbf{I} - \eta \mathbf{S}_j) \right) \mathbf{t}_l. \quad (\text{B.48})$$

As done in the proof of Theorem 5.6 (Appendix B.4.2), we will express (B.48) as one step of GD on F plus some noise. Of course, the noise terms here will be more complicated to handle than they were in Theorem 5.6. Due to summation by parts, the following identity holds:

$$\sum_{l=1}^{N/B} a_l b_l = a_{N/B} \sum_{j=1}^{N/B} b_j - \sum_{l=1}^{N/B-1} (a_{l+1} - a_l) \sum_{j=1}^l b_j.$$

We apply this to the last term of (B.48), by substituting $a_i = \prod_{j=N/B}^{l+1} (\mathbf{I} - \eta \mathbf{S}_j)$ and $b_l = \mathbf{t}_l$:

$$\eta \sum_{l=1}^{N/B} \left(\prod_{j=N/B}^{l+1} (\mathbf{I} - \eta \mathbf{S}_j) \right) \mathbf{t}_l = \eta \sum_{l=1}^{N/B} \mathbf{t}_l - \eta^2 \sum_{l=1}^{N/B-1} \left(\prod_{j=N/B}^{l+2} (\mathbf{I} - \eta \mathbf{S}_j) \right) \mathbf{S}_{l+1} \sum_{j=1}^l \mathbf{t}_j. \quad (\text{B.49})$$

We also apply the summation by parts to the inner summation of $\mathbf{t}_l$'s (B.46):

$$\begin{aligned} \mathbf{t}_l &:= \frac{1}{M} \sum_{m=1}^M \sum_{i=(l-1)B+1}^{lB} \left(\prod_{j=lB}^{i+1} (\mathbf{I} - \eta \mathbf{H}_j^m) \right) \nabla f_{\sigma_k^m(i)}^m(\mathbf{y}_{k,0}) \\ &= \frac{1}{M} \sum_{m=1}^M \sum_{i=(l-1)B+1}^{lB} \nabla f_{\sigma_k^m(i)}^m(\mathbf{y}_{k,0}) \\ &\quad - \frac{\eta}{M} \sum_{m=1}^M \sum_{i=(l-1)B+1}^{lB-1} \left(\prod_{t=lB}^{i+2} (\mathbf{I} - \eta \mathbf{H}_t^m) \right) \mathbf{H}_{i+1}^m \sum_{j=(l-1)B+1}^i \nabla f_{\sigma_k^m(j)}^m(\mathbf{y}_{k,0}) \end{aligned} \quad (\text{B.50})$$

Substituting (B.50) to (B.49) gives

$$\mathbf{y}_{k+1,0} = \mathbf{y}_{k,0} - \eta N \nabla F(\mathbf{y}_{k,0}) + \eta^2 \mathbf{r}_{k,1} + \eta^2 \mathbf{r}_{k,2} - \eta^3 \mathbf{r}_{k,3},$$

where $\mathbf{r}_{k,1}$, $\mathbf{r}_{k,2}$, and $\mathbf{r}_{k,3}$ are noise terms defined as

$$\begin{aligned} \mathbf{r}_{k,1} &:= \frac{1}{M} \sum_{l=1}^{N/B} \sum_{m=1}^M \sum_{i=(l-1)B+1}^{lB-1} \left(\prod_{t=lB}^{i+2} (\mathbf{I} - \eta \mathbf{H}_t^m) \right) \mathbf{H}_{i+1}^m \sum_{t=(l-1)B+1}^i \nabla f_{\sigma_k^m(t)}^m(\mathbf{y}_{k,0}), \\ \mathbf{r}_{k,2} &:= \frac{1}{M} \sum_{l=1}^{N/B-1} \left(\prod_{j=N/B}^{l+2} (\mathbf{I} - \eta \mathbf{S}_j) \right) \mathbf{S}_{l+1} \sum_{m=1}^M \sum_{t=1}^{lB} \nabla f_{\sigma_k^m(t)}^m(\mathbf{y}_{k,0}), \\ \mathbf{r}_{k,3} &:= \frac{1}{M} \sum_{l=1}^{N/B-1} \left(\prod_{j=N/B}^{l+2} (\mathbf{I} - \eta \mathbf{S}_j) \right) \mathbf{S}_{l+1} \times \\ &\quad \sum_{j=1}^l \sum_{m=1}^M \sum_{i=(j-1)B+1}^{jB-1} \left(\prod_{t=jB}^{i+2} (\mathbf{I} - \eta \mathbf{H}_t^m) \right) \mathbf{H}_{i+1}^m \sum_{t=(j-1)B+1}^i \nabla f_{\sigma_k^m(t)}^m(\mathbf{y}_{k,0}). \end{aligned}$$

Defining $\mathbf{r}_k := \mathbf{r}_{k,1} + \mathbf{r}_{k,2} - \eta \mathbf{r}_{k,3}$, it follows from L -smoothness of F that

$$\begin{aligned} &F(\mathbf{y}_{k+1,0}) - F(\mathbf{y}_{k,0}) \\ &\leq \langle \nabla F(\mathbf{y}_{k,0}), \mathbf{y}_{k+1,0} - \mathbf{y}_{k,0} \rangle + \frac{L}{2} \|\mathbf{y}_{k+1,0} - \mathbf{y}_{k,0}\|^2 \\ &\leq -\eta N \|\nabla F(\mathbf{y}_{k,0})\|^2 + \eta^2 \|\nabla F(\mathbf{y}_{k,0})\| \|\mathbf{r}_k\| + \frac{\eta^2 L}{2} \|N \nabla F(\mathbf{y}_{k,0}) + \eta \mathbf{r}_k\|^2 \\ &\leq (-\eta N + \eta^2 L N^2) \|\nabla F(\mathbf{y}_{k,0})\|^2 + \eta^2 \|\nabla F(\mathbf{y}_{k,0})\| \|\mathbf{r}_k\| + \eta^4 L \|\mathbf{r}_k\|^2. \end{aligned} \quad (\text{B.51})$$

Bounding noise terms using concentration. We next bound $\|\mathbf{r}_k\|$ by bounding each $\|\mathbf{r}_{k,1}\|$, $\|\mathbf{r}_{k,2}\|$, and $\|\mathbf{r}_{k,3}\|$. From this point on, we write $\mathbf{g}_t^m := \nabla f_{\sigma_k^m(t)}^m(\mathbf{y}_{k,0})$ to simplify notation.

$$\begin{aligned} \|\mathbf{r}_{k,1}\| &= \left\| \frac{1}{M} \sum_{l=1}^{N/B} \sum_{m=1}^M \sum_{i=(l-1)B+1}^{lB-1} \left(\prod_{t=lB}^{i+2} (\mathbf{I} - \eta \mathbf{H}_t^m) \right) \mathbf{H}_{i+1}^m \sum_{t=(l-1)B+1}^i \mathbf{g}_t^m \right\| \\ &\leq \frac{1}{M} \sum_{l=1}^{N/B} \sum_{m=1}^M \sum_{i=(l-1)B+1}^{lB-1} \left\| \left(\prod_{t=lB}^{i+2} (\mathbf{I} - \eta \mathbf{H}_t^m) \right) \mathbf{H}_{i+1}^m \sum_{t=(l-1)B+1}^i \mathbf{g}_t^m \right\| \\ &\leq \frac{L(1+\eta L)^B}{M} \sum_{l=1}^{N/B} \sum_{m=1}^M \sum_{i=(l-1)B+1}^{lB-1} \left\| \sum_{t=(l-1)B+1}^i \mathbf{g}_t^m \right\|, \end{aligned} \quad (\text{B.52})$$

where we used $\|\mathbf{H}_i^m\| \leq L$.

Recall from the theorem statement that $K \geq 7\rho\kappa \log(MNK^2)$ and $\eta = \frac{\log(MNK^2)}{\mu NK}$. Hence,

$$(1 + \eta L)^B = \left(1 + \frac{\kappa \log(MNK^2)}{NK}\right)^B \leq \left(1 + \frac{1}{7\rho N}\right)^B \leq \left(1 + \frac{1}{7B}\right)^B \leq e^{1/7}. \quad (\text{B.53})$$

Also note from the definition of $\mathbf{S}_l$ (B.47) that $\|\mathbf{S}_l\| \leq LB(1 + \eta L)^B \leq e^{1/7}LB$, which we use to get similar bounds for the next two terms $\mathbf{r}_{k,2}$ and $\mathbf{r}_{k,3}$.

$$\begin{aligned} \|\mathbf{r}_{k,2}\| &= \left\| \frac{1}{M} \sum_{l=1}^{N/B-1} \left(\prod_{j=N/B}^{l+2} (\mathbf{I} - \eta \mathbf{S}_j) \right) \mathbf{S}_{l+1} \sum_{m=1}^M \sum_{t=1}^{lB} \mathbf{g}_t^m \right\| \\ &\leq \frac{e^{1/7}LB(1 + e^{1/7}\eta LB)^{N/B}}{M} \sum_{l=1}^{N/B-1} \left\| \sum_{m=1}^M \sum_{t=1}^{lB} \mathbf{g}_t^m \right\|, \end{aligned} \quad (\text{B.54})$$

and we can bound

$$\begin{aligned} (1 + e^{1/7}\eta LB)^{N/B} &= \left(1 + \frac{e^{1/7}\kappa B \log(MNK^2)}{NK}\right)^{N/B} \\ &\leq \left(1 + \frac{e^{1/7}B}{7\rho N}\right)^{N/B} \leq \exp\left(\frac{e^{1/7}}{7}\right). \end{aligned} \quad (\text{B.55})$$

We similarly bound the norm of the last noise term $\mathbf{r}_{k,3}$:

$$\begin{aligned} \|\mathbf{r}_{k,3}\| &\leq \frac{e^{1/7}LB(1 + e^{1/7}\eta LB)^{N/B}}{M} \times \\ &\quad \sum_{l=1}^{N/B-1} \left\| \sum_{j=1}^l \sum_{m=1}^M \sum_{i=(j-1)B+1}^{jB-1} \left(\prod_{t=jB}^{i+2} (\mathbf{I} - \eta \mathbf{H}_t^m) \right) \mathbf{H}_{i+1}^m \sum_{t=(j-1)B+1}^i \mathbf{g}_t^m \right\| \\ &\leq \frac{e^{1/7}L^2B(1 + e^{1/7}\eta LB)^{N/B}(1 + \eta L)^B}{M} \sum_{l=1}^{N/B-1} \sum_{j=1}^l \sum_{m=1}^M \sum_{i=(j-1)B+1}^{jB-1} \left\| \sum_{t=(j-1)B+1}^i \mathbf{g}_t^m \right\| \\ &\leq \frac{11L^2B}{7M} \sum_{l=1}^{N/B-1} \sum_{j=1}^l \sum_{m=1}^M \sum_{i=(j-1)B+1}^{jB-1} \left\| \sum_{t=(j-1)B+1}^i \mathbf{g}_t^m \right\|, \end{aligned} \quad (\text{B.56})$$

where the last inequality used (B.53), (B.55), and $e^{2/7} \exp(e^{1/7}/7) \leq 11/7$.

Given the bounds (B.52), (B.54), and (B.56), we now use Lemma B.1 to get high-probability bounds for the partial sums of $\mathbf{g}_t^m$ that appear in the bounds. For any i satisfying $(j-1)B + 1 \leq i \leq jB - 1$, where $j \in [N/B]$, and for any $m \in [M]$, the following bound holds with

probability at least $1 - \frac{\delta}{2MNK}$:

$$\begin{aligned} & \left\| \frac{1}{i - (j-1)B} \sum_{t=(l-1)B+1}^i \mathbf{g}_t^m - \frac{1}{N} \sum_{t=1}^N \mathbf{g}_t^m \right\| \\ &= \left\| \frac{1}{i - (j-1)B} \sum_{t=(l-1)B+1}^i \nabla f_{\sigma_k^m(t)}^m(\mathbf{y}_{k,0}) - \nabla F^m(\mathbf{y}_{k,0}) \right\| \leq \nu \sqrt{\frac{8 \log \frac{4MNK}{\delta}}{i - (j-1)B}}. \end{aligned}$$

From this, with probability at least $1 - \frac{\delta}{2MNK}$ we have

$$\left\| \sum_{t=(j-1)B+1}^i \mathbf{g}_t^m \right\| \leq \nu \sqrt{8(i - (j-1)B) \log \frac{4MNK}{\delta}} + (i - (j-1)B) \|\nabla F^m(\mathbf{y}_{k,0})\|. \quad (\text{B.57})$$

Similarly, for $l \in [N/B - 1]$, the following bound holds with probability at least $1 - \frac{B\delta}{2NK}$:

$$\begin{aligned} & \left\| \frac{1}{lBM} \sum_{m=1}^M \sum_{t=1}^{lB} \mathbf{g}_t^m - \frac{1}{MN} \sum_{m=1}^M \sum_{t=1}^N \mathbf{g}_t^m \right\| \\ &= \left\| \frac{1}{lBM} \sum_{m=1}^M \sum_{t=1}^{lB} \nabla f_{\sigma_k^m(t)}^m(\mathbf{y}_{k,0}) - \nabla F(\mathbf{y}_{k,0}) \right\| \leq \nu \sqrt{\frac{8 \log \frac{4NK}{B\delta}}{lBM}}, \end{aligned}$$

which gives us

$$\left\| \sum_{m=1}^M \sum_{t=1}^{lB} \mathbf{g}_t^m \right\| \leq \nu \sqrt{8lBM \log \frac{4NK}{B\delta}} + lBM \|\nabla F(\mathbf{y}_{k,0})\| \quad (\text{B.58})$$

By applying the union bound, with probability at least $1 - \frac{\delta}{K}$, the bound (B.57) holds for all $m \in [M]$ and $i \in \cup_{j=1}^{N/B} [(j-1)B + 1 : jB - 1]$, and the bound (B.58) holds for all $l \in [N/B - 1]$.

We now substitute the bounds (B.57) and (B.58) to (B.52), (B.54), and (B.56) to get upper bounds for $\|\mathbf{r}_{k,1}\|$, $\|\mathbf{r}_{k,2}\|$, and $\|\mathbf{r}_{k,3}\|$, respectively. First,

$$\begin{aligned} & \|\mathbf{r}_{k,1}\| \\ & \leq \frac{e^{1/7}L}{M} \sum_{l=1}^{N/B} \sum_{m=1}^M \sum_{i=(l-1)B+1}^{lB-1} \left(\nu \sqrt{8(i - (l-1)B) \log \frac{4MNK}{\delta}} \right. \\ & \quad \left. + (i - (l-1)B) \|\nabla F^m(\mathbf{y}_{k,0})\| \right) \\ & = \frac{e^{1/7}\sqrt{8}L\nu N}{B} \left(\sum_{i=1}^{B-1} \sqrt{i} \right) \sqrt{\log \frac{4MNK}{\delta}} + \frac{e^{1/7}LN}{B} \left(\sum_{i=1}^{B-1} i \right) \left(\frac{1}{M} \sum_{m=1}^M \|\nabla F^m(\mathbf{y}_{k,0})\| \right) \end{aligned}$$

$$\leq \frac{24L\nu N(B^{3/2} - 1)}{11B} \sqrt{\log \frac{4MNK}{\delta}} + \frac{3LN(B-1)}{5} (\tau + \rho \|\nabla F(\mathbf{y}_{k,0})\|), \quad (\text{B.59})$$

where the last inequality used $\sum_{i=1}^{B-1} \sqrt{i} \leq \int_1^B \sqrt{z} dz = \frac{2}{3}(B^{3/2} - 1)$ and Assumption 5.3. For the next noise term, we have $e^{1/7}(1 + e^{1/7}\eta LB)^{N/B} \leq 7/5$, so

$$\begin{aligned} \|\mathbf{r}_{k,2}\| &\leq \frac{7LB}{5M} \sum_{l=1}^{N/B-1} \left(\nu \sqrt{8lBM \log \frac{4NK}{B\delta}} + lBM \|\nabla F(\mathbf{y}_{k,0})\| \right) \\ &= \frac{7\sqrt{8}L\nu B^{3/2}}{5M^{1/2}} \left(\sum_{l=1}^{N/B-1} \sqrt{l} \right) \sqrt{\log \frac{4NK}{B\delta}} + \frac{7LB^2}{5} \left(\sum_{l=1}^{N/B-1} l \right) \|\nabla F(\mathbf{y}_{k,0})\| \\ &\leq \frac{8L\nu(N^{3/2} - B^{3/2})}{3M^{1/2}} \sqrt{\log \frac{4NK}{B\delta}} + \frac{7LN(N-B)}{10} \|\nabla F(\mathbf{y}_{k,0})\|. \end{aligned} \quad (\text{B.60})$$

Lastly,

$$\begin{aligned} \|\mathbf{r}_{k,3}\| &\leq \frac{11L^2B}{7M} \sum_{l=1}^{N/B-1} \sum_{j=1}^l \sum_{m=1}^M \sum_{i=(j-1)B+1}^{jB-1} \left(\nu \sqrt{8(i-(j-1)B) \log \frac{4MNK}{\delta}} \right. \\ &\quad \left. + (i-(j-1)B) \|\nabla F^m(\mathbf{y}_{k,0})\| \right) \\ &= \frac{11\sqrt{8}L^2\nu B}{7} \left(\sum_{l=1}^{N/B-1} l \right) \left(\sum_{i=1}^{B-1} \sqrt{i} \right) \sqrt{\log \frac{4MNK}{\delta}} \\ &\quad + \frac{11L^2B}{7} \left(\sum_{l=1}^{N/B-1} l \right) \left(\sum_{i=1}^{B-1} i \right) \left(\frac{1}{M} \sum_{m=1}^M \|\nabla F^m(\mathbf{y}_{k,0})\| \right) \\ &\leq \frac{3L^2\nu N(N-B)(B^{3/2} - 1)}{2B} \sqrt{\log \frac{4MNK}{\delta}} \\ &\quad + \frac{2L^2N(N-B)(B-1)}{5} (\tau + \rho \|\nabla F(\mathbf{y}_{k,0})\|). \end{aligned} \quad (\text{B.61})$$

Recalling the definition $\mathbf{r}_k := \mathbf{r}_{k,1} + \mathbf{r}_{k,2} - \eta \mathbf{r}_{k,3}$, we get an upper bound for $\|\mathbf{r}_k\|$ from (B.59), (B.60), and (B.61):

$$\begin{aligned} \|\mathbf{r}_k\| &\leq \|\mathbf{r}_{k,1}\| + \|\mathbf{r}_{k,2}\| + \eta \|\mathbf{r}_{k,3}\| \\ &\leq L\nu \sqrt{\log \frac{4MNK}{\delta}} \left(\frac{24N(B^{3/2} - 1)}{11B} + \frac{8(N^{3/2} - B^{3/2})}{3M^{1/2}} \right. \\ &\quad \left. + \frac{3\eta LN(N-B)(B^{3/2} - 1)}{2B} \right) \\ &\quad + L\tau \left(\frac{3N(B-1)}{5} + \frac{2\eta LN(N-B)(B-1)}{5} \right) \end{aligned}$$

$$+ L \|\nabla F(\mathbf{y}_{k,0})\| \left(\frac{3\rho N(B-1)}{5} + \frac{7N(N-B)}{10} + \frac{2\eta L\rho N(N-B)(B-1)}{5} \right). \quad (\text{B.62})$$

Recall again that we have $K \geq 7\rho\kappa \log(MNK^2)$ and $\eta = \frac{\log(MNK^2)}{\mu NK}$, so $\eta LN \leq 1/7$. Using this and $N - B \leq N$, we can further simplify (B.62).

$$\begin{aligned} \|\mathbf{r}_k\| &\leq L\nu \sqrt{\log \frac{4MNK}{\delta}} \underbrace{\left(\frac{12N(B^{3/2}-1)}{5B} + \frac{8(N^{3/2}-B^{3/2})}{3M^{1/2}} \right)}_{=:\Phi} + \frac{2L\tau N(B-1)}{3} \\ &\quad + L \|\nabla F(\mathbf{y}_{k,0})\| \left(\frac{2\rho N(B-1)}{3} + \frac{7N(N-B)}{10} \right) \\ &\leq L\nu\Phi \sqrt{\log \frac{4MNK}{\delta}} + \frac{2L\tau N(B-1)}{3} + \frac{7L\rho N^2}{5} \|\nabla F(\mathbf{y}_{k,0})\|, \end{aligned} \quad (\text{B.63})$$

which holds with probability at least $1 - \frac{\delta}{K}$. The bound (B.63) holds for all $k \in [K]$ with probability $1 - \delta$ if we apply the union bound over $k = 1, \dots, K$. Next, by $(a + b + c)^2 \leq 3a^2 + 3b^2 + 3c^2$, we have

$$\|\mathbf{r}_k\|^2 \leq 3L^2\nu^2\Phi^2 \log \frac{4MNK}{\delta} + \frac{4L^2\tau^2 N^2(B-1)^2}{3} + \frac{147L^2\rho^2 N^4}{25} \|\nabla F(\mathbf{y}_{k,0})\|^2, \quad (\text{B.64})$$

which also holds for all $k \in [K]$ with probability at least $1 - \delta$.

Getting a high-probability convergence rate. Given our high-probability bounds (B.63) and (B.64), we can substitute them to (B.51) and get

$$\begin{aligned} &F(\mathbf{y}_{k+1,0}) - F(\mathbf{y}_{k,0}) \\ &\leq (-\eta N + \eta^2 LN^2) \|\nabla F(\mathbf{y}_{k,0})\|^2 + \eta^2 \|\nabla F(\mathbf{y}_{k,0})\| \|\mathbf{r}_k\| + \eta^4 L \|\mathbf{r}_k\|^2 \\ &\leq \left(-\eta N + \eta^2 LN^2 + \frac{7\eta^2 L\rho N^2}{5} + \frac{147\eta^4 L^3 \rho^2 N^4}{25} \right) \|\nabla F(\mathbf{y}_{k,0})\|^2 \\ &\quad + \eta^2 L\nu\Phi \sqrt{\log \frac{4MNK}{\delta}} \|\nabla F(\mathbf{y}_{k,0})\| + 3\eta^4 L^3 \nu^2 \Phi^2 \log \frac{4MNK}{\delta} \\ &\quad + \frac{2\eta^2 L\tau N(B-1)}{3} \|\nabla F(\mathbf{y}_{k,0})\| + \frac{4\eta^4 L^3 \tau^2 N^2(B-1)^2}{3}. \end{aligned} \quad (\text{B.65})$$

The following terms in the RHS of (B.65) can be bounded using $ab \leq \frac{a^2}{2} + \frac{b^2}{2}$:

$$\eta^2 L\nu\Phi \sqrt{\log \frac{4MNK}{\delta}} \|\nabla F(\mathbf{y}_{k,0})\|$$

$$\begin{aligned}
&= \left(\frac{\eta^{1/2} N^{1/2}}{\sqrt{8}} \|\nabla F(\mathbf{y}_{k,0})\| \right) \left(\frac{\sqrt{8} \eta^{3/2} L \nu \Phi}{N^{1/2}} \sqrt{\log \frac{4MNK}{\delta}} \right) \\
&\leq \frac{\eta N}{16} \|\nabla F(\mathbf{y}_{k,0})\|^2 + \frac{4\eta^3 L^2 \nu^2 \Phi^2}{N} \log \frac{4MNK}{\delta},
\end{aligned} \tag{B.66}$$

$$\begin{aligned}
&\frac{2\eta^2 L \tau N (B-1)}{3} \|\nabla F(\mathbf{y}_{k,0})\| \\
&= \left(\frac{\eta^{1/2} N^{1/2}}{\sqrt{8}} \|\nabla F(\mathbf{y}_{k,0})\| \right) \left(\frac{2\sqrt{8} \eta^{3/2} L \tau N^{1/2} (B-1)}{3} \right) \\
&\leq \frac{\eta N}{16} \|\nabla F(\mathbf{y}_{k,0})\|^2 + \frac{16\eta^3 L^2 \tau^2 N (B-1)^2}{9}.
\end{aligned} \tag{B.67}$$

Substituting (B.66) and (B.67) to (B.65) results in

$$\begin{aligned}
&F(\mathbf{y}_{k+1,0}) - F(\mathbf{y}_{k,0}) \\
&\leq \left(-\frac{7}{8} \eta N + \eta^2 L N^2 + \frac{7\eta^2 L \rho N^2}{5} + \frac{147\eta^4 L^3 \rho^2 N^4}{25} \right) \|\nabla F(\mathbf{y}_{k,0})\|^2 \\
&\quad + \frac{\eta^3 L^2 \nu^2 (4 + 3\eta L N) \Phi^2}{N} \log \frac{4MNK}{\delta} + \frac{4\eta^3 L^2 \tau^2 (4 + 3\eta L N) N (B-1)^2}{9}.
\end{aligned} \tag{B.68}$$

Again, we have $K \geq 7\rho\kappa \log(MNK^2)$ and $\eta = \frac{\log(MNK^2)}{\mu NK}$, so $\eta L \rho N \leq \frac{1}{7}$. Since the inequality $-\frac{7}{8}z + \frac{12}{5}z^2 + \frac{147}{25}z^4 \leq -\frac{1}{2}z$ holds on $z \in [0, \frac{1}{7}]$, we have

$$\begin{aligned}
&-\frac{7}{8} \eta N + \eta^2 L N^2 + \frac{7\eta^2 L \rho N^2}{5} + \frac{147\eta^4 L^3 \rho^2 N^4}{25} \\
&\leq -\frac{7}{8} \eta N + \frac{12\eta^2 L \rho N^2}{5} + \frac{147\eta^4 L^3 \rho^3 N^4}{25} \leq -\frac{1}{2} \eta N.
\end{aligned}$$

Substituting this inequality to (B.68), together with $4 + 3\eta L N \leq \frac{31}{7} < \frac{9}{2}$, yields

$$\begin{aligned}
F(\mathbf{y}_{k+1,0}) - F(\mathbf{y}_{k,0}) &\leq -\frac{\eta N}{2} \|\nabla F(\mathbf{y}_{k,0})\|^2 + \frac{9\eta^3 L^2 \nu^2 \Phi^2}{2N} \log \frac{4MNK}{\delta} \\
&\quad + 2\eta^3 L^2 \tau^2 N (B-1)^2.
\end{aligned}$$

We now recall that F is μ -PL, so $\|\nabla F(\mathbf{y}_{k,0})\|^2 \geq 2\mu(F(\mathbf{y}_{k,0}) - F^*)$:

$$\begin{aligned}
F(\mathbf{y}_{k+1,0}) - F^* &\leq (1 - \eta\mu N)(F(\mathbf{y}_{k,0}) - F^*) \\
&\quad + \frac{9\eta^3 L^2 \nu^2 \Phi^2}{2N} \log \frac{4MNK}{\delta} + 2\eta^3 L^2 \tau^2 N (B-1)^2.
\end{aligned} \tag{B.69}$$

Recall that (B.69) holds for all $k \in [K]$, with probability $1 - \delta$. Therefore, by unrolling the inequality,

$$\begin{aligned}
F(\mathbf{y}_{K, \frac{N}{B}}) - F^* &\leq (1 - \eta\mu N)^K (F(\mathbf{y}_0) - F^*) \\
&\quad + \left(\frac{9\eta^3 L^2 \nu^2 \Phi^2}{2N} \log \frac{4MNK}{\delta} + 2\eta^3 L^2 \tau^2 N(B-1)^2 \right) \sum_{k=0}^{K-1} (1 - \eta\mu N)^k \\
&\leq (1 - \eta\mu N)^K (F(\mathbf{y}_0) - F^*) \\
&\quad + \frac{9\eta^2 L^2 \nu^2 \Phi^2}{2\mu N^2} \log \frac{4MNK}{\delta} + \frac{2\eta^2 L^2 \tau^2 (B-1)^2}{\mu}.
\end{aligned} \tag{B.70}$$

Recall that $\Phi := \frac{12N(B^{3/2}-1)}{5B} + \frac{8(N^{3/2}-B^{3/2})}{3M^{1/2}}$, hence

$$\Phi^2 \leq \frac{288N^2(B^{3/2}-1)^2}{25B^2} + \frac{128(N^{3/2}-B^{3/2})^2}{9M}.$$

Substituting this inequality and also $\eta = \frac{\log(MNK^2)}{\mu NK}$ gives

$$\begin{aligned}
&F(\mathbf{y}_{K, \frac{N}{B}}) - F^* \\
&\leq \frac{F(\mathbf{y}_0) - F^*}{MNK^2} + \frac{2L^2 \tau^2 (B-1)^2}{\mu^3 N^2 K^2} \log^2(MNK^2) \\
&\quad + \frac{9L^2 \nu^2}{2\mu^3 N^4 K^2} \log \frac{4MNK}{\delta} \log^2(MNK^2) \left(\frac{288N^2(B^{3/2}-1)^2}{25B^2} + \frac{128(N^{3/2}-B^{3/2})^2}{9M} \right) \\
&= \frac{F(\mathbf{y}_0) - F^*}{MNK^2} + \tilde{O}\left(\frac{L^2 \tau^2}{\mu^3} \frac{B^2}{N^2 K^2}\right) + \tilde{O}\left(\frac{L^2 \nu^2}{\mu^3} \frac{B}{N^2 K^2}\right) + \tilde{O}\left(\frac{L^2 \nu^2}{\mu^3} \frac{1}{MNK^2}\right),
\end{aligned}$$

with probability at least $1 - \delta$. This finishes the proof.

B.4.4 Proof of upper bound for minibatch RR with SYNC SHUF (Theorem 5.13)

The first part (“One epoch as one step of GD plus noise”) of the proof is identical to that of Theorem 5.6. We start from the second part.

Bounding noise term using concentration. It is left to bound $\|\mathbf{r}_k\|$. As seen in (B.30), we have

$$\|\mathbf{r}_k\| \leq LB(1 + \eta LB)^{N/B} \sum_{i=1}^{N/B-1} \left\| \sum_{j=1}^i \mathbf{g}_j \right\|. \tag{B.71}$$

Recall from the theorem statement that $K \geq 6\kappa \log(MNK^2)$ and $\eta = \frac{\log(MNK^2)}{\mu NK}$. This means that

$$(1 + \eta LB)^{N/B} = \left(1 + \frac{\kappa B \log(MNK^2)}{NK}\right)^{N/B} \leq \left(1 + \frac{B}{6N}\right)^{N/B} \leq e^{1/6}. \quad (\text{B.72})$$

Next, we bound the norm of

$$\sum_{j=1}^i \mathbf{g}_j = \frac{1}{M} \sum_{m=1}^M \sum_{j=1}^{iB} \nabla f_{\sigma_k^m(j)}^m(\mathbf{x}_{k,0}), \quad (\text{B.73})$$

exploiting our modification SYNC SHUF as well as Lemma B.1. For each $\nabla f_{\sigma_k^m(j)}^m(\mathbf{x}_{k,0})$, we first add and subtract its corresponding $\nabla \bar{f}_{\sigma_k^m(j)}^m(\mathbf{x}_{k,0})$, where $\bar{f}_i := \frac{1}{M} \sum_{m=1}^M f_i^m$ as defined in Assumption 5.4. This way, (B.73) can be decomposed into two sums $\sum_{j=1}^i \mathbf{g}_j = \frac{1}{M}(\mathbf{p}_i + \mathbf{q}_i)$, where

$$\begin{aligned} \mathbf{p}_i &:= \sum_{m=1}^M \sum_{j=1}^{iB} \nabla \bar{f}_{\sigma_k^m(j)}^m(\mathbf{x}_{k,0}), \\ \mathbf{q}_i &:= \sum_{m=1}^M \sum_{j=1}^{iB} \nabla f_{\sigma_k^m(j)}^m(\mathbf{x}_{k,0}) - \nabla \bar{f}_{\sigma_k^m(j)}^m(\mathbf{x}_{k,0}). \end{aligned}$$

Using this decomposition, we will derive high-probability bounds for $\|\mathbf{p}_i\|$ and $\|\mathbf{q}_i\|$.

To simplify expressions to follow, we decompose iBM (i.e., the total number of component gradients that are summed up) into a multiple of N and the remainder. Let

$$\alpha(i) := \left\lfloor \frac{iBM}{N} \right\rfloor, \quad \beta(i) := iBM - N\alpha(i),$$

so that iBM is decomposed into $N\alpha(i)$ and the remainder $0 \leq \beta(i) < N$. Using this new notation, we can write $\mathbf{p}_i$ as

$$\mathbf{p}_i = \sum_{m=1}^M \sum_{j=1}^{\frac{N\alpha(i)}{M}} \nabla \bar{f}_{\sigma_k^m(j)}^m(\mathbf{x}_{k,0}) + \sum_{m=1}^M \sum_{j=\frac{N\alpha(i)}{M}+1}^{iB} \nabla \bar{f}_{\sigma_k^m(j)}^m(\mathbf{x}_{k,0}). \quad (\text{B.74})$$

Here, recall that with SYNC SHUF, we defined $\sigma_k^m(j) := \sigma((j + \frac{N}{M}\pi(m)) \bmod N)$. With this choice of “shifted” permutations, one can notice that $\{\sigma_k^m(j)\}_{m=1, j=1}^{M, \frac{N}{M}} = [N]$, meaning that adding $\bar{f}_{\sigma_k^m(j)}^m$ for $m \in [M]$ and $j \in [N/M]$ results in the sum of all N $\bar{f}_i$ ’s. In fact, this happens if we sum over $m \in [M]$ and any N/M consecutive j ’s. From this observation and

$F = \frac{1}{N} \sum_{i=1}^N \bar{f}_i$, (B.74) can be written as

$$\mathbf{p}_i = N\alpha(i)\nabla F(\mathbf{x}_{k,0}) + \sum_{m=1}^M \sum_{j=\frac{N\alpha(i)}{M}+1}^{iB} \nabla \bar{f}_{\sigma((j+\frac{Nm}{M}) \bmod N)}(\mathbf{x}_{k,0}). \quad (\text{B.75})$$

Assume for now that $\beta(i) > 0$, i.e., $N\alpha(i) < iBM$. The summation in the second term of RHS in (B.75) is a without-replacement sum (note that the indices $j + \frac{Nm}{M}$ do not overlap) of $\beta(i)$ terms. Hence, it is equal in distribution to $\sum_{j=1}^{\beta(i)} \nabla \bar{f}_{\sigma(j)}(\mathbf{x}_{k,0})$. Also, from Assumption 5.2, it can be easily checked that for any $i \in [N]$

$$\left\| \nabla \bar{f}_i(\mathbf{x}_{k,0}) - \nabla F(\mathbf{x}_{k,0}) \right\| \leq \nu.$$

These observations mean that we can apply Lemma B.1 to get a concentration bound, with $M \leftarrow 1$, $\mathbf{v}_i^1 \leftarrow \nabla \bar{f}_i(\mathbf{x}_{k,0})$, $n \leftarrow \beta(i)$, and $\delta \leftarrow \frac{B\delta}{2NK}$. By Lemma B.1, with probability at least $1 - \frac{B\delta}{2NK}$ (over the randomness in σ), we have

$$\left\| \frac{1}{\beta(i)} \sum_{m=1}^M \sum_{j=\frac{N\alpha(i)}{M}+1}^{iB} \nabla \bar{f}_{\sigma((j+\frac{Nm}{M}) \bmod N)}(\mathbf{x}_{k,0}) - \nabla F(\mathbf{x}_{k,0}) \right\| \leq \nu \sqrt{\frac{8 \log \frac{4NK}{B\delta}}{\beta(i)}}. \quad (\text{B.76})$$

Combining (B.75) and (B.76), we get the following upper bound on $\|\mathbf{p}_i\|$, which holds with probability at least $1 - \frac{B\delta}{2NK}$.

$$\begin{aligned} \|\mathbf{p}_i\| &\leq \|iBM\nabla F(\mathbf{x}_{k,0})\| + \left\| \sum_{m=1}^M \sum_{j=\frac{N\alpha(i)}{M}+1}^{iB} \nabla \bar{f}_{\sigma((j+\frac{Nm}{M}) \bmod N)}(\mathbf{x}_{k,0}) - \beta(i)\nabla F(\mathbf{x}_{k,0}) \right\| \\ &\leq iBM \|\nabla F(\mathbf{x}_{k,0})\| + \nu \sqrt{8\beta(i) \log \frac{4NK}{B\delta}} \\ &\leq iBM \|\nabla F(\mathbf{x}_{k,0})\| + \nu \sqrt{N} \sqrt{8 \log \frac{4NK}{B\delta}}, \end{aligned} \quad (\text{B.77})$$

where the last inequality used $\beta(i) < N$. Also recall that we assumed $\beta(i) > 0$ in order to use Lemma B.1 and derive (B.77). However, note that even with $\beta(i) = 0$, the bound (B.77) trivially holds.

We next bound $\|\mathbf{q}_i\|$. This time, we will apply Lemma B.1 to the permutation π over the local machines. To do this, we will condition on a fixed instantiation of the permutation σ and derive a high-probability bound that holds with conditional probability at least $1 - \frac{B\delta}{2NK}$. The conditional probability is at least $1 - \frac{B\delta}{2NK}$ irrespective of the choice of σ , so we can conclude that the (unconditional) probability that our bound holds is also at least

$$1 - \frac{B\delta}{2NK}.$$

Without loss of generality, choose the instantiation $\sigma(l) = l$ for all $l \in [N]$. With this σ , we have $\sigma_k^m(j) := (j + \frac{N}{M}\pi(m)) \bmod N$, so the vector $\mathbf{q}_i$ reads

$$\mathbf{q}_i = \sum_{m=1}^M \sum_{j=1}^{iB} \nabla f_{(j + \frac{N}{M}\pi(m)) \bmod N}^m(\mathbf{x}_{k,0}) - \nabla \bar{f}_{(j + \frac{N}{M}\pi(m)) \bmod N}(\mathbf{x}_{k,0}). \quad (\text{B.78})$$

Let us consider rewriting this summation as the sum over $l \in [N]$, where l appears in the subscript of the component functions. One can check that

$$l = \left(j + \frac{N}{M}\pi(m)\right) \bmod N \iff \frac{N}{M} \mid (l - j) \text{ and } \pi(m) = (l - j) \frac{M}{N} \bmod M,$$

where $a \mid b$ denotes “ a divides b .” From this, we can rewrite (B.78) as

$$\mathbf{q}_i = \sum_{l=1}^N \underbrace{\sum_{\substack{j \in [iB] \\ \frac{N}{M} \mid (l-j)}} \left(\nabla f_l^{\pi^{-1}((l-j)\frac{M}{N} \bmod M)}(\mathbf{x}_{k,0}) - \nabla \bar{f}_l(\mathbf{x}_{k,0}) \right)}_{=: \mathbf{q}_{i,l}}. \quad (\text{B.79})$$

By the same reasoning above and below (B.75), we can see from (B.79) that for a given index $l \in [N]$, the cardinality of the set $\mathcal{J}_l := \{j \in [iB] : \frac{N}{M} \mid (l - j)\}$ is either $\alpha(i)$ or $\alpha(i) + 1$. From this, we notice that each $\mathbf{q}_{i,l}$ is a without-replacement sum of $\alpha(i)$ or $\alpha(i) + 1$ terms. For now, suppose $\alpha(i) > 0$. For each $\mathbf{q}_{i,l}$, we can apply Lemma B.1 to it, and show that with probability (conditioned on the instantiation $\sigma(l) = l$) at least $1 - \frac{B\delta}{2N^2K}$, we have

$$\|\mathbf{q}_{i,l}\| \leq \begin{cases} \lambda \sqrt{8(\alpha(i)) \log \frac{4N^2K}{B\delta}} & \text{if } |\mathcal{J}_l| = \alpha(i), \\ \lambda \sqrt{8(\alpha(i) + 1) \log \frac{4N^2K}{B\delta}} & \text{if } |\mathcal{J}_l| = \alpha(i) + 1. \end{cases}$$

Note that cases in the RHS are all bounded from above by $\lambda \sqrt{16\alpha(i) \log \frac{4N^2K}{B\delta}}$. Applying union bound on all $l \in [N]$, we get that with probability at least $1 - \frac{B\delta}{2NK}$, we have

$$\|\mathbf{q}_i\| \leq \lambda N \sqrt{16\alpha(i) \log \frac{4N^2K}{B\delta}} \leq \lambda N \sqrt{\frac{16iBM}{N} \log \frac{4N^2K}{B\delta}} = 4\lambda \sqrt{iBMN \log \frac{4N^2K}{B\delta}}. \quad (\text{B.80})$$

Now consider the case $\alpha(i) = 0$. Recall from the definition $\alpha(i) := \left\lfloor \frac{iBM}{N} \right\rfloor$ that $\alpha(i) = 0$ implies $iBM < N$. In this case, $\mathbf{q}_{i,l} = \mathbf{0}$ for $N - iBM$ indices l satisfying $|\mathcal{J}_l| = 0$, and $\|\mathbf{q}_{i,l}\| \leq \lambda$ for the remaining iBM l 's satisfying $|\mathcal{J}_l| = 1$. Summing up, $\|\mathbf{q}_i\|$ is bounded from above by λiBM , which is in fact less than the upper bound in (B.80). Therefore, the

bound (B.80) holds even for $\alpha(i) = 0$.

Recall that our goal was to find a bound on the norm of $\sum_{j=1}^i \mathbf{g}_j = \frac{1}{M}(\mathbf{p}_i + \mathbf{q}_i)$. From the high-probability bounds obtained in (B.77) and (B.80), with probability at least $1 - \frac{B\delta}{NK}$,

$$\left\| \sum_{j=1}^i \mathbf{g}_j \right\| \leq iB \|\nabla F(\mathbf{x}_{k,0})\| + \frac{\nu\sqrt{N}}{M} \sqrt{8 \log \frac{4NK}{B\delta}} + 4\lambda \sqrt{\frac{iBN}{M}} \sqrt{\log \frac{4N^2K}{B\delta}}. \quad (\text{B.81})$$

We can now substitute (B.72) and (B.81) to (B.71) to get

$$\begin{aligned} & \|\mathbf{r}_k\| \\ & \leq e^{1/6} LB \sum_{i=1}^{N/B-1} \left(iB \|\nabla F(\mathbf{x}_{k,0})\| + \frac{\nu\sqrt{N}}{M} \sqrt{8 \log \frac{4NK}{B\delta}} + 4\lambda \sqrt{\frac{iBN}{M}} \sqrt{\log \frac{4N^2K}{B\delta}} \right) \\ & \leq e^{1/6} LB^2 \|\nabla F(\mathbf{x}_{k,0})\| \sum_{i=1}^{N/B-1} i + \frac{e^{1/6} \sqrt{8} L \nu N^{1/2} (N-B)}{M} \sqrt{\log \frac{4NK}{B\delta}} \\ & \quad + \frac{4e^{1/6} L \lambda B^{3/2} N^{1/2}}{M^{1/2}} \int_1^{N/B} \sqrt{t} dt \sqrt{\log \frac{4N^2K}{B\delta}} \\ & \leq LN(N-B) \|\nabla F(\mathbf{x}_{k,0})\| + \frac{7L\nu N^{1/2} (N-B)}{2M} \sqrt{\log \frac{4NK}{B\delta}} \\ & \quad + \frac{7L\lambda N^{1/2} (N^{3/2} - B^{3/2})}{2M^{1/2}} \sqrt{\log \frac{4N^2K}{B\delta}}, \end{aligned} \quad (\text{B.82})$$

which holds with probability at least $1 - \frac{\delta}{K}$, due to the union bound over $i = 1, \dots, N/B - 1$. The bound (B.82) holds for all $k \in [K]$ with probability $1 - \delta$ if we apply the union bound over $k = 1, \dots, K$. Next, by $(a + b + c)^2 \leq 3a^2 + 3b^2 + 3c^2$, we have

$$\begin{aligned} \|\mathbf{r}_k\|^2 & \leq 3L^2 N^2 (N-B)^2 \|\nabla F(\mathbf{x}_{k,0})\|^2 + \frac{147L^2 \nu^2 N (N-B)^2}{4M^2} \log \frac{4NK}{B\delta} \\ & \quad + \frac{147L^2 \lambda^2 N (N^{3/2} - B^{3/2})^2}{4M} \log \frac{4N^2K}{B\delta}, \end{aligned} \quad (\text{B.83})$$

which also holds for all $k \in [K]$ with probability at least $1 - \delta$.

Getting a high-probability convergence rate. Given our high-probability bounds (B.82) and (B.83), we can substitute them to (B.29) and get

$$\begin{aligned} & F(\mathbf{x}_{k+1,0}) - F(\mathbf{x}_{k,0}) \\ & \leq (-\eta N + \eta^2 L N^2) \|\nabla F(\mathbf{x}_{k,0})\|^2 + \eta^2 \|\nabla F(\mathbf{x}_{k,0})\| \|\mathbf{r}_k\| + \eta^4 L \|\mathbf{r}_k\|^2 \\ & \leq \left(-\eta N + \eta^2 L N^2 + \eta^2 L N (N-B) + 3\eta^4 L^3 N^2 (N-B)^2 \right) \|\nabla F(\mathbf{x}_{k,0})\|^2 \end{aligned}$$

$$\begin{aligned}
& + \frac{7\eta^2 L \nu N^{1/2} (N - B)}{2M} \sqrt{\log \frac{4NK}{B\delta}} \|\nabla F(\mathbf{x}_{k,0})\| + \frac{147\eta^4 L^3 \nu^2 N (N - B)^2}{4M^2} \log \frac{4NK}{B\delta} \\
& + \frac{7\eta^2 L \lambda N^{1/2} (N^{3/2} - B^{3/2})}{2M^{1/2}} \sqrt{\log \frac{4N^2 K}{B\delta}} \|\nabla F(\mathbf{x}_{k,0})\| \\
& + \frac{147\eta^4 L^3 \lambda^2 N (N^{3/2} - B^{3/2})^2}{4M} \log \frac{4N^2 K}{B\delta}.
\end{aligned} \tag{B.84}$$

Two terms in the RHS of (B.84) can be bounded using $ab \leq \frac{a^2}{2} + \frac{b^2}{2}$:

$$\begin{aligned}
& \frac{7\eta^2 L \nu N^{1/2} (N - B)}{2M} \sqrt{\log \frac{4NK}{B\delta}} \|\nabla F(\mathbf{x}_{k,0})\| \\
& = \left(\frac{\eta^{1/2} N^{1/2}}{2\sqrt{2}} \|\nabla F(\mathbf{x}_{k,0})\| \right) \left(\frac{7\sqrt{2}\eta^{3/2} L \nu (N - B)}{M} \sqrt{\log \frac{4NK}{B\delta}} \right) \\
& \leq \frac{\eta N}{16} \|\nabla F(\mathbf{x}_{k,0})\|^2 + \frac{49\eta^3 L^2 \nu^2 (N - B)^2}{M^2} \log \frac{4NK}{B\delta},
\end{aligned} \tag{B.85}$$

$$\begin{aligned}
& \frac{7\eta^2 L \lambda N^{1/2} (N^{3/2} - B^{3/2})}{2M^{1/2}} \sqrt{\log \frac{4N^2 K}{B\delta}} \|\nabla F(\mathbf{x}_{k,0})\| \\
& = \left(\frac{\eta^{1/2} N^{1/2}}{2\sqrt{2}} \|\nabla F(\mathbf{x}_{k,0})\| \right) \left(\frac{7\sqrt{2}\eta^{3/2} L \lambda (N^{3/2} - B^{3/2})}{M^{1/2}} \sqrt{\log \frac{4N^2 K}{B\delta}} \right) \\
& \leq \frac{\eta N}{16} \|\nabla F(\mathbf{x}_{k,0})\|^2 + \frac{49\eta^3 L^2 \lambda^2 (N^{3/2} - B^{3/2})^2}{M} \log \frac{4N^2 K}{B\delta},
\end{aligned} \tag{B.86}$$

Putting inequalities (B.85) and (B.86) to (B.84) and noting $N - B \leq N$ gives

$$\begin{aligned}
F(\mathbf{x}_{k+1,0}) - F(\mathbf{x}_{k,0}) & \leq \left(-\frac{7}{8}\eta N + 2\eta^2 L N^2 + 3\eta^4 L^3 N^4 \right) \|\nabla F(\mathbf{x}_{k,0})\|^2 \\
& + \frac{49\eta^3 L^2 (4 + 3\eta L N) \nu^2 (N - B)^2}{4M^2} \log \frac{4NK}{B\delta} \\
& + \frac{49\eta^3 L^2 (4 + 3\eta L N) \lambda^2 (N^{3/2} - B^{3/2})^2}{4M} \log \frac{4N^2 K}{B\delta}.
\end{aligned} \tag{B.87}$$

Recall from $K \geq 6\kappa \log(M^2 N K^2)$ and $\eta = \frac{\log(M^2 N K^2)}{\mu N K}$ that $\eta L N \leq \frac{1}{6}$. Since the inequality $-\frac{7}{8}z + 2z^2 + 3z^4 \leq -\frac{1}{2}z$ holds on $z \in [0, \frac{1}{6}]$, we have

$$-\frac{7}{8}\eta N + 2\eta^2 L N^2 + 3\eta^4 L^3 N^4 \leq -\frac{1}{2}\eta N.$$

Applying this bound to (B.87) results in

$$F(\mathbf{x}_{k+1,0}) - F(\mathbf{x}_{k,0}) \leq -\frac{\eta N}{2} \|\nabla F(\mathbf{x}_{k,0})\|^2 + \frac{56\eta^3 L^2 \nu^2 (N - B)^2}{M^2} \log \frac{4NK}{B\delta}$$

$$+ \frac{56\eta^3 L^2 \lambda^2 (N^{3/2} - B^{3/2})^2}{M} \log \frac{4N^2 K}{B\delta}$$

We now recall that F is μ -PL, so $\|\nabla F(\mathbf{x}_{k,0})\|^2 \geq 2\mu(F(\mathbf{x}_{k,0}) - F^*)$:

$$\begin{aligned} F(\mathbf{x}_{k+1,0}) - F^* &\leq (1 - \eta\mu N)(F(\mathbf{x}_{k,0}) - F^*) + \frac{56\eta^3 L^2 \nu^2 (N - B)^2}{M^2} \log \frac{4NK}{B\delta} \\ &\quad + \frac{56\eta^3 L^2 \lambda^2 (N^{3/2} - B^{3/2})^2}{M} \log \frac{4N^2 K}{B\delta} \end{aligned} \quad (\text{B.88})$$

Recall that (B.88) holds for all $k \in [K]$, with probability $1 - \delta$. Therefore, by unrolling the inequality, and using $\sum_{k=0}^{K-1} (1 - \eta\mu N)^k \leq \frac{1}{\eta\mu N}$, we get

$$\begin{aligned} F(\mathbf{x}_{K, \frac{N}{B}}) - F^* &\leq (1 - \eta\mu N)^K (F(\mathbf{x}_0) - F^*) + \frac{56\eta^2 L^2 \nu^2 (N - B)^2}{\mu M^2 N} \log \frac{4NK}{B\delta} \\ &\quad + \frac{56\eta^2 L^2 \lambda^2 (N^{3/2} - B^{3/2})^2}{\mu MN} \log \frac{4N^2 K}{B\delta} \end{aligned} \quad (\text{B.89})$$

Lastly, substituting $\eta = \frac{\log(M^2 NK^2)}{\mu NK}$ to (B.89) gives

$$\begin{aligned} F(\mathbf{x}_{K, \frac{N}{B}}) - F^* &\leq \frac{F(\mathbf{x}_0) - F^*}{M^2 NK^2} + \frac{56L^2 \nu^2 (N - B)^2 \log \frac{4NK}{B\delta} \log^2(M^2 NK^2)}{\mu^3 M^2 N^3 K^2} \\ &\quad + \frac{56L^2 \lambda^2 (N^{3/2} - B^{3/2})^2 \log \frac{4N^2 K}{B\delta} \log^2(M^2 NK^2)}{\mu^3 MN^3 K^2} \\ &= \frac{F(\mathbf{x}_0) - F^*}{M^2 NK^2} + \tilde{\mathcal{O}} \left(\frac{L^2}{\mu^3} \left(\frac{\nu^2}{M^2 NK^2} + \frac{\lambda^2}{MK^2} \right) \right). \end{aligned}$$

B.4.5 Proof of upper bound for local RR with SYNC SHUF (Theorem 5.14)

The first part (“One epoch as one step of GD plus noise”) of the proof is identical to that of Theorem 5.7. The first part defines our “noise” $\mathbf{r}_k$ as the sum of three terms $\mathbf{r}_k := \mathbf{r}_{k,1} + \mathbf{r}_{k,2} - \eta \mathbf{r}_{k,3}$. We start from the second part.

Bounding noise terms using concentration. We next bound $\|\mathbf{r}_k\|$ by bounding each $\|\mathbf{r}_{k,1}\|$, $\|\mathbf{r}_{k,2}\|$, and $\|\mathbf{r}_{k,3}\|$. We have already seen from (B.52), (B.53), (B.54), (B.55), and (B.56) in Appendix B.4.3 that

$$\|\mathbf{r}_{k,1}\| \leq \frac{e^{1/7} L}{M} \sum_{l=1}^{N/B} \sum_{m=1}^M \sum_{i=(l-1)B+1}^{lB-1} \left\| \sum_{t=(l-1)B+1}^i \nabla f_{\sigma_k^m(t)}^m(\mathbf{y}_{k,0}) \right\|, \quad (\text{B.90})$$

$$\|\mathbf{r}_{k,2}\| \leq \frac{7LB}{5M} \sum_{l=1}^{N/B-1} \left\| \sum_{m=1}^M \sum_{t=1}^{lB} \nabla f_{\sigma_k^m(t)}^m(\mathbf{y}_{k,0}) \right\|, \quad (\text{B.91})$$

$$\|\mathbf{r}_{k,3}\| \leq \frac{11L^2B}{7M} \sum_{l=1}^{N/B-1} \sum_{j=1}^l \sum_{m=1}^M \sum_{i=(j-1)B+1}^{jB-1} \left\| \sum_{t=(j-1)B+1}^i \nabla f_{\sigma_k^m(t)}^m(\mathbf{y}_{k,0}) \right\|. \quad (\text{B.92})$$

As in the previous subsections, the key is to bound the norm of the partial sums of $\nabla f_{\sigma_k^m(t)}^m(\mathbf{y}_{k,0})$ using Lemma B.1. For the summations appearing in (B.90) and (B.92), we apply Lemma B.1 in the same way as (B.57). For any i satisfying $(j-1)B+1 \leq i \leq jB-1$, where $j \in [N/B]$, and for any $m \in [M]$, the following bound holds with probability at least $1 - \frac{\delta}{3MNK}$:

$$\begin{aligned} \left\| \sum_{t=(j-1)B+1}^i \nabla f_{\sigma_k^m(t)}^m(\mathbf{y}_{k,0}) \right\| &\leq \nu \sqrt{8(i - (j-1)B) \log \frac{6MNK}{\delta}} \\ &\quad + (i - (j-1)B) \|\nabla F^m(\mathbf{y}_{k,0})\|. \end{aligned} \quad (\text{B.93})$$

For the summation that appear in (B.91), we use the techniques from Appendix B.4.4. For each $l \in [N/B-1]$, we can similarly decompose $\sum_{m=1}^M \sum_{t=1}^{lB} \nabla f_{\sigma_k^m(t)}^m(\mathbf{y}_{k,0})$ into $\mathbf{p}_l + \mathbf{q}_l$, where

$$\begin{aligned} \mathbf{p}_l &:= \sum_{m=1}^M \sum_{t=1}^{lB} \nabla \bar{f}_{\sigma_k^m(t)}^m(\mathbf{y}_{k,0}), \\ \mathbf{q}_l &:= \sum_{m=1}^M \sum_{t=1}^{lB} \nabla f_{\sigma_k^m(t)}^m(\mathbf{y}_{k,0}) - \nabla \bar{f}_{\sigma_k^m(t)}^m(\mathbf{y}_{k,0}). \end{aligned}$$

As done in Appendix B.4.4, we can follow the same steps and show a high-probability bound, which is a slightly different version of (B.77): with probability at least $1 - \frac{B\delta}{3NK}$,

$$\|\mathbf{p}_l\| \leq lBM \|\nabla F(\mathbf{y}_{k,0})\| + \nu \sqrt{N} \sqrt{8 \log \frac{6NK}{B\delta}}. \quad (\text{B.94})$$

Similarly, for $\|\mathbf{q}_l\|$ we can show a slight modification of (B.80):

$$\|\mathbf{q}_l\| \leq 4\lambda \sqrt{lBMN \log \frac{6N^2K}{B\delta}}, \quad (\text{B.95})$$

which holds with probability at least $1 - \frac{B\delta}{3NK}$. Combining (B.94) and (B.95), with probability at least $1 - \frac{2B\delta}{3NK}$, we have

$$\left\| \sum_{m=1}^M \sum_{t=1}^{lB} \nabla f_{\sigma_k^m(t)}^m(\mathbf{y}_{k,0}) \right\| \leq lBM \|\nabla F(\mathbf{y}_{k,0})\| + \nu \sqrt{N} \sqrt{8 \log \frac{6NK}{B\delta}}$$

$$+ 4\lambda \sqrt{lBMN \log \frac{6N^2K}{B\delta}}. \quad (\text{B.96})$$

By applying the union bound, with probability at least $1 - \frac{\delta}{K}$, the bound (B.93) holds for all $m \in [M]$ and $i \in \cup_{j=1}^{N/B} [(j-1)B+1 : jB-1]$, and the bound (B.96) holds for all $l \in [N/B-1]$.

We now substitute the bounds (B.93) and (B.96) to (B.90), (B.91), and (B.92) to get upper bounds for $\|\mathbf{r}_{k,1}\|$, $\|\mathbf{r}_{k,2}\|$, and $\|\mathbf{r}_{k,3}\|$, respectively. For $\|\mathbf{r}_{k,1}\|$ and $\|\mathbf{r}_{k,3}\|$, we can apply the same calculations as in (B.59) and (B.61), modulo the fact that Assumption 5.3 is now implied by Assumption 5.4, with constants $\tau = \lambda$ and $\rho = 1$. We obtain

$$\|\mathbf{r}_{k,1}\| \leq \frac{24L\nu N(B^{3/2}-1)}{11B} \sqrt{\log \frac{6MNK}{\delta}} + \frac{3LN(B-1)}{5} (\lambda + \|\nabla F(\mathbf{y}_{k,0})\|), \quad (\text{B.97})$$

$$\begin{aligned} \|\mathbf{r}_{k,3}\| &\leq \frac{3L^2\nu N(N-B)(B^{3/2}-1)}{2B} \sqrt{\log \frac{6MNK}{\delta}} \\ &\quad + \frac{2L^2N(N-B)(B-1)}{5} (\lambda + \|\nabla F(\mathbf{y}_{k,0})\|). \end{aligned} \quad (\text{B.98})$$

For $\|\mathbf{r}_{k,2}\|$, we have

$$\begin{aligned} \|\mathbf{r}_{k,2}\| &\leq \frac{7LB}{5M} \sum_{l=1}^{N/B-1} \left(lBM \|\nabla F(\mathbf{y}_{k,0})\| + \nu\sqrt{N} \sqrt{8 \log \frac{6NK}{B\delta}} + 4\lambda \sqrt{lBMN \log \frac{6N^2K}{B\delta}} \right) \\ &= \frac{7LB^2}{5} \left(\sum_{l=1}^{N/B-1} l \right) \|\nabla F(\mathbf{y}_{k,0})\| + \frac{7\sqrt{8}L\nu N^{1/2}(N-B)}{5M} \sqrt{\log \frac{6NK}{B\delta}} \\ &\quad + \frac{28L\lambda B^{3/2}N^{1/2}}{5M^{1/2}} \left(\sum_{l=1}^{N/B-1} \sqrt{l} \right) \sqrt{\log \frac{6N^2K}{B\delta}} \\ &\leq \frac{7LN(N-B)}{10} \|\nabla F(\mathbf{y}_{k,0})\| + \frac{4L\nu N^{1/2}(N-B)}{M} \sqrt{\log \frac{6NK}{B\delta}} \\ &\quad + \frac{15L\lambda N^{1/2}(N^{3/2}-B^{3/2})}{4M^{1/2}} \sqrt{\log \frac{6N^2K}{B\delta}}. \end{aligned} \quad (\text{B.99})$$

Recalling the definition $\mathbf{r}_k := \mathbf{r}_{k,1} + \mathbf{r}_{k,2} - \eta \mathbf{r}_{k,3}$, we get an upper bound for $\|\mathbf{r}_k\|$ from (B.97), (B.98), and (B.99):

$$\begin{aligned} \|\mathbf{r}_k\| &\leq \|\mathbf{r}_{k,1}\| + \|\mathbf{r}_{k,2}\| + \eta \|\mathbf{r}_{k,3}\| \end{aligned}$$

$$\begin{aligned}
&\leq L\nu\sqrt{\log\frac{6MNK}{\delta}}\left(\frac{24N(B^{3/2}-1)}{11B}+\frac{4N^{1/2}(N-B)}{M}\right. \\
&\quad \left.+\frac{3\eta LN(N-B)(B^{3/2}-1)}{2B}\right) \\
&\quad +L\lambda\left(\frac{3N(B-1)}{5}+\frac{15N^{1/2}(N^{3/2}-B^{3/2})}{4M^{1/2}}\sqrt{\log\frac{6N^2K}{B\delta}}\right. \\
&\quad \left.+\frac{2\eta LN(N-B)(B-1)}{5}\right) \\
&\quad +L\|\nabla F(\mathbf{y}_{k,0})\|\left(\frac{3N(B-1)}{5}+\frac{7N(N-B)}{10}+\frac{2\eta LN(N-B)(B-1)}{5}\right). \quad (\text{B.100})
\end{aligned}$$

Recall again that we have $K \geq 7\kappa \log(M^2NK^2)$ and $\eta = \frac{\log(M^2NK^2)}{\mu NK}$, so $\eta LN \leq 1/7$. Using this and $N - B \leq N$, we can further simplify (B.100).

$$\begin{aligned}
\|\mathbf{r}_k\| &\leq L\nu\sqrt{\log\frac{6MNK}{\delta}}\underbrace{\left(\frac{12N(B^{3/2}-1)}{5B}+\frac{4N^{1/2}(N-B)}{M}\right)}_{=:\Phi_\nu} \\
&\quad +L\lambda\sqrt{\log\frac{6N^2K}{B\delta}}\underbrace{\left(\frac{2N(B-1)}{3}+\frac{15N^{1/2}(N^{3/2}-B^{3/2})}{4M^{1/2}}\right)}_{=:\Phi_\lambda} \\
&\quad +L\|\nabla F(\mathbf{y}_{k,0})\|\left(\frac{2N(B-1)}{3}+\frac{7N(N-B)}{10}\right) \\
&\leq L\nu\Phi_\nu\sqrt{\log\frac{6MNK}{\delta}}+L\lambda\Phi_\lambda\sqrt{\log\frac{6N^2K}{B\delta}}+\frac{7LN^2}{5}\|\nabla F(\mathbf{y}_{k,0})\|, \quad (\text{B.101})
\end{aligned}$$

which holds with probability at least $1 - \frac{\delta}{K}$. The bound (B.101) holds for all $k \in [K]$ with probability $1 - \delta$ if we apply the union bound over $k = 1, \dots, K$. Next, by $(a + b + c)^2 \leq 3a^2 + 3b^2 + 3c^2$, we have

$$\|\mathbf{r}_k\|^2 \leq 3L^2\nu^2\Phi_\nu^2\log\frac{6MNK}{\delta}+3L^2\lambda^2\Phi_\lambda^2\log\frac{6N^2K}{B\delta}+\frac{147L^2N^4}{25}\|\nabla F(\mathbf{y}_{k,0})\|^2, \quad (\text{B.102})$$

which also holds for all $k \in [K]$ with probability at least $1 - \delta$.

Getting a high-probability convergence rate. Given our high-probability bounds (B.101) and (B.102), we can substitute them to (B.51) and get

$$\begin{aligned}
&F(\mathbf{y}_{k+1,0}) - F(\mathbf{y}_{k,0}) \\
&\leq (-\eta N + \eta^2 LN^2)\|\nabla F(\mathbf{y}_{k,0})\|^2 + \eta^2\|\nabla F(\mathbf{y}_{k,0})\|\|\mathbf{r}_k\| + \eta^4 L\|\mathbf{r}_k\|^2
\end{aligned}$$

$$\begin{aligned}
&\leq \left(-\eta N + \eta^2 L N^2 + \frac{7\eta^2 L N^2}{5} + \frac{147\eta^4 L^3 N^4}{25} \right) \|\nabla F(\mathbf{y}_{k,0})\|^2 \\
&\quad + \eta^2 L \nu \Phi_\nu \sqrt{\log \frac{6MNK}{\delta}} \|\nabla F(\mathbf{y}_{k,0})\| + 3\eta^4 L^3 \nu^2 \Phi_\nu^2 \log \frac{6MNK}{\delta} \\
&\quad + \eta^2 L \lambda \Phi_\lambda \sqrt{\log \frac{6N^2 K}{B\delta}} \|\nabla F(\mathbf{y}_{k,0})\| + 3\eta^4 L^3 \lambda^2 \Phi_\lambda^2 \log \frac{6N^2 K}{B\delta}. \tag{B.103}
\end{aligned}$$

The following terms in the RHS of (B.103) can be bounded using $ab \leq \frac{a^2}{2} + \frac{b^2}{2}$:

$$\begin{aligned}
&\eta^2 L \nu \Phi_\nu \sqrt{\log \frac{6MNK}{\delta}} \|\nabla F(\mathbf{y}_{k,0})\| \\
&= \left(\frac{\eta^{1/2} N^{1/2}}{\sqrt{8}} \|\nabla F(\mathbf{y}_{k,0})\| \right) \left(\frac{\sqrt{8} \eta^{3/2} L \nu \Phi_\nu}{N^{1/2}} \sqrt{\log \frac{6MNK}{\delta}} \right) \\
&\leq \frac{\eta N}{16} \|\nabla F(\mathbf{y}_{k,0})\|^2 + \frac{4\eta^3 L^2 \nu^2 \Phi_\nu^2}{N} \log \frac{6MNK}{\delta}, \tag{B.104}
\end{aligned}$$

$$\begin{aligned}
&\eta^2 L \lambda \Phi_\lambda \sqrt{\log \frac{6N^2 K}{B\delta}} \|\nabla F(\mathbf{y}_{k,0})\| \\
&= \left(\frac{\eta^{1/2} N^{1/2}}{\sqrt{8}} \|\nabla F(\mathbf{y}_{k,0})\| \right) \left(\frac{\sqrt{8} \eta^{3/2} L \lambda \Phi_\lambda}{N^{1/2}} \sqrt{\log \frac{6N^2 K}{B\delta}} \right) \\
&\leq \frac{\eta N}{16} \|\nabla F(\mathbf{y}_{k,0})\|^2 + \frac{4\eta^3 L^2 \lambda^2 \Phi_\lambda^2}{N} \log \frac{6N^2 K}{B\delta}. \tag{B.105}
\end{aligned}$$

Substituting (B.104) and (B.105) to (B.103) results in

$$\begin{aligned}
&F(\mathbf{y}_{k+1,0}) - F(\mathbf{y}_{k,0}) \\
&\leq \left(-\frac{7}{8}\eta N + \frac{12\eta^2 L N^2}{5} + \frac{147\eta^4 L^3 \rho^2 N^4}{25} \right) \|\nabla F(\mathbf{y}_{k,0})\|^2 \\
&\quad + \frac{\eta^3 L^2 \nu^2 (4 + 3\eta L N) \Phi_\nu^2}{N} \log \frac{6MNK}{\delta} + \frac{\eta^3 L^2 \lambda^2 (4 + 3\eta L N) \Phi_\lambda^2}{N} \log \frac{6N^2 K}{B\delta}. \tag{B.106}
\end{aligned}$$

Again, we have $K \geq 7\kappa \log(M^2 N K^2)$ and $\eta = \frac{\log(M^2 N K^2)}{\mu N K}$, so $\eta L N \leq \frac{1}{7}$. Since the inequality $-\frac{7}{8}z + \frac{12}{5}z^2 + \frac{147}{25}z^4 \leq -\frac{1}{2}z$ holds on $z \in [0, \frac{1}{7}]$, we have

$$-\frac{7}{8}\eta N + \frac{12\eta^2 L \rho N^2}{5} + \frac{147\eta^4 L^3 \rho^3 N^4}{25} \leq -\frac{1}{2}\eta N.$$

Substituting this inequality to (B.106), together with $4 + 3\eta L N \leq \frac{31}{7} < \frac{9}{2}$, yields

$$F(\mathbf{y}_{k+1,0}) - F(\mathbf{y}_{k,0}) \leq -\frac{\eta N}{2} \|\nabla F(\mathbf{y}_{k,0})\|^2 + \frac{9\eta^3 L^2 \nu^2 \Phi_\nu^2}{2N} \log \frac{6MNK}{\delta}$$

$$+ \frac{9\eta^3 L^2 \lambda^2 \Phi_\lambda^2}{2N} \log \frac{6N^2 K}{B\delta}.$$

We now recall that F is μ -PL, so $\|\nabla F(\mathbf{y}_{k,0})\|^2 \geq 2\mu(F(\mathbf{y}_{k,0}) - F^*)$:

$$\begin{aligned} F(\mathbf{y}_{k+1,0}) - F^* &\leq (1 - \eta\mu N)(F(\mathbf{y}_{k,0}) - F^*) + \frac{9\eta^3 L^2 \nu^2 \Phi_\nu^2}{2N} \log \frac{6MNK}{\delta} \\ &\quad + \frac{9\eta^3 L^2 \lambda^2 \Phi_\lambda^2}{2N} \log \frac{6N^2 K}{B\delta}. \end{aligned} \quad (\text{B.107})$$

Recall that (B.107) holds for all $k \in [K]$, with probability $1 - \delta$. Therefore, by unrolling the inequality, and using $\sum_{k=0}^{K-1} (1 - \eta\mu N)^k \leq \frac{1}{\eta\mu N}$, we get

$$\begin{aligned} F(\mathbf{y}_{K, \frac{N}{B}}) - F^* &\leq (1 - \eta\mu N)^K (F(\mathbf{y}_0) - F^*) + \frac{9\eta^2 L^2 \nu^2 \Phi_\nu^2}{2\mu N^2} \log \frac{6MNK}{\delta} \\ &\quad + \frac{9\eta^2 L^2 \lambda^2 \Phi_\lambda^2}{2\mu N^2} \log \frac{6N^2 K}{B\delta}. \end{aligned} \quad (\text{B.108})$$

Recall that $\Phi_\nu := \frac{12N(B^{3/2}-1)}{5B} + \frac{4N^{1/2}(N-B)}{M}$ and $\Phi_\lambda := \frac{2N(B-1)}{3} + \frac{15N^{1/2}(N^{3/2}-B^{3/2})}{4M^{1/2}}$, hence

$$\begin{aligned} \Phi_\nu^2 &\leq \frac{288N^2(B^{3/2}-1)^2}{25B^2} + \frac{32N(N-B)^2}{M^2}, \\ \Phi_\lambda^2 &\leq \frac{8N^2(B-1)^2}{9} + \frac{225N(N^{3/2}-B^{3/2})^2}{2M}. \end{aligned}$$

Substituting these inequalities and also $\eta = \frac{\log(M^2 N K^2)}{\mu N K}$ gives

$$\begin{aligned} &F(\mathbf{y}_{K, \frac{N}{B}}) - F^* \\ &\leq \frac{F(\mathbf{y}_0) - F^*}{M^2 N K^2} \\ &\quad + \frac{9L^2 \nu^2 \log \frac{6MNK}{\delta} \log^2(M^2 N K^2)}{2\mu^3 N^4 K^2} \left(\frac{288N^2(B^{3/2}-1)^2}{25B^2} + \frac{32N(N-B)^2}{M^2} \right) \\ &\quad + \frac{9L^2 \lambda^2 \log \frac{6N^2 K}{B\delta} \log^2(M^2 N K^2)}{2\mu^3 N^4 K^2} \left(\frac{8N^2(B-1)^2}{9} + \frac{225N(N^{3/2}-B^{3/2})^2}{2M} \right) \\ &= \frac{F(\mathbf{y}_0) - F^*}{M^2 N K^2} + \tilde{\mathcal{O}} \left(\frac{L^2}{\mu^3} \left(\frac{\nu^2 B}{N^2 K^2} + \frac{\nu^2}{M^2 N K^2} + \frac{\lambda^2 B^2}{N^2 K^2} + \frac{\lambda^2}{M K^2} \right) \right), \end{aligned}$$

with probability at least $1 - \delta$. This finishes the proof.

B.4.6 A generalized vector-valued Hoeffding-Serfling inequality

We extend the vector-valued Hoeffding-Serfling inequality proved in Schneider (2016) to account for the mean of multiple independent without-replacement sums.

Lemma B.1. *Suppose there are MN vectors $\{\mathbf{v}_i^m\}_{m=1, i=1}^{M, N} \in \mathbb{R}^d$ that satisfy $\|\mathbf{v}_i^m - \bar{\mathbf{v}}^m\| \leq \nu$ for $m \in [M]$, where $\bar{\mathbf{v}}^m := \frac{1}{N} \sum_{i=1}^N \mathbf{v}_i^m$. Consider M independently and uniformly sampled permutations $\sigma_1, \dots, \sigma_M \sim \text{Unif}(\mathcal{S}_N)$. For any $n \leq N - 1$, with probability at least $1 - \delta$, we have*

$$\left\| \frac{1}{Mn} \sum_{m=1}^M \sum_{i=1}^n \mathbf{v}_{\sigma_m(i)}^m - \frac{1}{M} \sum_{m=1}^M \bar{\mathbf{v}}^m \right\| \leq \nu \sqrt{\frac{8(1 - \frac{n-1}{N}) \log \frac{2}{\delta}}{Mn}}. \quad (\text{B.109})$$

Proof. The proof is an extension of Theorem 2 of Schneider (2016) which proves the $M = 1$ case for vectors in smooth separable Banach spaces. We prove our extended concentration inequality for $\mathbb{R}^d$, but we note that the proof technique can be applied directly to general smooth separable Banach spaces, as done in Schneider (2016). Below, we state a special case of Theorem 3 of Pinelis (1992) and Theorem 3.5 of Pinelis (1994), because this $\mathbb{R}^d$ case serves our purpose.

Lemma B.2 (Pinelis (1992, 1994)). *Suppose that a sequence of random variables $\{\mathbf{x}_j\}_{j \geq 0}$ is a martingale taking values in $\mathbb{R}^d$, and $\sum_{j=1}^{\infty} \text{ess sup} \|\mathbf{x}_j - \mathbf{x}_{j-1}\|^2 \leq c^2$ for some $c > 0$. Then, for $\lambda > 0$,*

$$\mathbb{P}\left(\sup\{\|\mathbf{x}_j\| : j \geq 0\} \geq \lambda\right) \leq 2 \exp\left(-\frac{\lambda^2}{2c^2}\right).$$

The proof of Lemma B.1 proceeds by defining a sequence of random variables $\{\mathbf{x}_j\}$, showing that it is a martingale, and applying Lemma B.2 to prove our concentration bound. For $m \in [M]$, define index functions $k_m : \mathbb{N} \cup \{0\} \rightarrow [0 : n]$ in the following way:

$$\begin{aligned} k_m(j) &:= \max\{0, \min\{n, j - (m-1)n\}\} \\ &= \begin{cases} 0 & \text{if } j \leq (m-1)n, \\ j - (m-1)n & \text{if } (m-1)n + 1 \leq j \leq mn, \\ n & \text{if } j \geq mn + 1. \end{cases} \end{aligned}$$

Using these index functions, we introduce the following sequence of random variables $\{\mathbf{x}_j\}$:

$$\mathbf{x}_j := \sum_{m=1}^M \frac{1}{N - k_m(j)} \sum_{i=1}^{k_m(j)} (\mathbf{v}_{\sigma_m(i)}^m - \bar{\mathbf{v}}^m),$$

and show that this is a martingale, i.e.,

$$\mathbb{E}[\mathbf{x}_j \mid \mathbf{x}_1, \dots, \mathbf{x}_{j-1}] = \mathbf{x}_{j-1} \quad (\text{B.110})$$

for all $j \geq 1$. Notice first that by definition of k_m 's we have $\mathbf{x}_{Mn} = \mathbf{x}_{Mn+1} = \mathbf{x}_{Mn+2} = \dots$, so (B.110) is trivially satisfied for all $j > Mn$. Next, for any j satisfying $(l-1)n+1 \leq j \leq ln$ where $l \in [M]$, we have

$$\begin{aligned} \mathbf{x}_j &= \sum_{m=1}^{l-1} \frac{1}{N-n} \sum_{i=1}^n (\mathbf{v}_{\sigma_m(i)}^m - \bar{\mathbf{v}}^m) + \frac{1}{N-k_l(j)} \sum_{i=1}^{k_l(j)} (\mathbf{v}_{\sigma_l(i)}^l - \bar{\mathbf{v}}^l) \\ &= \mathbf{x}_{j-1} + \left(\frac{1}{N-k_l(j)} - \frac{1}{N-k_l(j)+1} \right) \sum_{i=1}^{k_l(j)-1} (\mathbf{v}_{\sigma_l(i)}^l - \bar{\mathbf{v}}^l) \\ &\quad + \frac{1}{N-k_l(j)} (\mathbf{v}_{\sigma_l(k_l(j))}^l - \bar{\mathbf{v}}^l) \end{aligned} \quad (\text{B.111})$$

Now note that for any $k \in [N-1]$, we have

$$\begin{aligned} \mathbb{E}[\mathbf{v}_{\sigma_l(k)}^l - \bar{\mathbf{v}}^l \mid \sigma_l(1), \dots, \sigma_l(k-1)] &= \frac{1}{N-k+1} \sum_{i=k}^N (\mathbf{v}_{\sigma_l(i)}^l - \bar{\mathbf{v}}^l) \\ &= -\frac{1}{N-k+1} \sum_{i=1}^{k-1} (\mathbf{v}_{\sigma_l(i)}^l - \bar{\mathbf{v}}^l), \end{aligned}$$

where the last equality used $\sum_{i=1}^N (\mathbf{v}_{\sigma_l(i)}^l - \bar{\mathbf{v}}^l) = \mathbf{0}$. Using applying this fact to (B.111) and noting $\frac{1}{N-k_l(j)} - \frac{1}{N-k_l(j)+1} = \frac{1}{(N-k_l(j))(N-k_l(j)+1)}$,

$$\begin{aligned} \mathbb{E}[\mathbf{x}_j \mid \mathbf{x}_1, \dots, \mathbf{x}_{j-1}] &= \mathbf{x}_{j-1} + \frac{1}{(N-k_l(j))(N-k_l(j)+1)} \sum_{i=1}^{k_l(j)-1} (\mathbf{v}_{\sigma_l(i)}^l - \bar{\mathbf{v}}^l) \\ &\quad + \frac{1}{N-k_l(j)} \mathbb{E}[\mathbf{v}_{\sigma_l(k_l(j))}^l - \bar{\mathbf{v}}^l \mid \mathbf{x}_1, \dots, \mathbf{x}_{j-1}] \\ &= \mathbf{x}_{j-1}, \end{aligned}$$

hence proving that $\{\mathbf{x}_j\}_{j \geq 0}$ is a martingale. We now apply Lemma B.2 to our $\{\mathbf{x}_j\}$. For j such that $(l-1)n+1 \leq j \leq ln$, notice from (B.111) that

$$\begin{aligned} (N-k_l(j))(\mathbf{x}_j - \mathbf{x}_{j-1}) &= \frac{1}{(N-k_l(j)+1)} \left[\sum_{i=1}^{k_l(j)-1} (\mathbf{v}_{\sigma_l(i)}^l - \bar{\mathbf{v}}^l) \right] + (\mathbf{v}_{\sigma_l(k_l(j))}^l - \bar{\mathbf{v}}^l) \\ &= -\frac{1}{(N-k_l(j)+1)} \left[\sum_{i=k_l(j)}^N (\mathbf{v}_{\sigma_l(i)}^l - \bar{\mathbf{v}}^l) \right] + (\mathbf{v}_{\sigma_l(k_l(j))}^l - \bar{\mathbf{v}}^l), \end{aligned}$$

which leads to

$$(N - k_l(j)) \|\mathbf{x}_j - \mathbf{x}_{j-1}\| \leq \frac{\nu \min\{k_l(j) - 1, N - k_l(j) + 1\}}{N - k_l(j) + 1} + \nu \leq 2\nu,$$

by the triangle inequality. From this, we get the bound c^2 in the statement of Lemma B.2:

$$\begin{aligned} \sum_{j=1}^{\infty} \text{ess sup} \|\mathbf{x}_j - \mathbf{x}_{j-1}\|^2 &= \sum_{j=1}^{Mn} \text{ess sup} \|\mathbf{x}_j - \mathbf{x}_{j-1}\|^2 \\ &\leq M \sum_{k=1}^n \frac{4\nu^2}{(N - k)^2} \\ &= \frac{4\nu^2 M}{(N - n)^2} + 4\nu^2 M \sum_{k=N-n+1}^{N-1} \frac{1}{k^2} \\ &\leq \frac{4\nu^2 M}{(N - n)^2} + \frac{4\nu^2 M(n-1)}{(N - n)N} \\ &= \frac{4\nu^2 Mn}{(N - n)^2} \left(1 - \frac{n-1}{N}\right), \end{aligned}$$

where the second inequality used the inequality that $\sum_{k=a+1}^b \frac{1}{k^2} \leq \frac{b-a}{a(b+1)}$ (Serfling, 1974, Lemma 2.1). Now, applying Lemma B.2 to $\{\mathbf{x}_j\}$ with $c^2 = \frac{4\nu^2 Mn}{(N-n)^2} \left(1 - \frac{n-1}{N}\right)$ gives

$$\mathbb{P}(\|\mathbf{x}_{Mn}\| \geq \lambda) \leq \mathbb{P}(\sup\{\|\mathbf{x}_j\| : j \geq 0\} \geq \lambda) \leq 2 \exp\left(-\frac{\lambda^2(N-n)^2}{8\nu^2 Mn \left(1 - \frac{n-1}{N}\right)}\right). \quad (\text{B.112})$$

Recall from the definition of $\{\mathbf{x}_j\}$ that

$$\mathbf{x}_{Mn} = \frac{1}{N-n} \sum_{m=1}^M \sum_{i=1}^n (\mathbf{v}_{\sigma_m(i)}^m - \bar{\mathbf{v}}^m).$$

Substituting $\lambda = \frac{Mn\epsilon}{N-n}$ to (B.112) gives

$$\mathbb{P}\left(\left\|\frac{1}{Mn} \sum_{m=1}^M \sum_{i=1}^n (\mathbf{v}_{\sigma_m(i)}^m - \bar{\mathbf{v}}^m)\right\| \geq \epsilon\right) \leq 2 \exp\left(-\frac{Mn\epsilon^2}{8\nu^2 \left(1 - \frac{n-1}{N}\right)}\right),$$

which finishes the proof. $\square$

B.4.7 How can we avoid uniform bounds over $\mathbb{R}^d$ in our assumptions?

In Section 5.2, we introduced Assumptions 5.2, 5.3, and 5.4 on the intra- and inter-machine deviation. The assumptions required that inequalities such as

$\|\nabla f_i^m(\mathbf{x}) - \nabla F^m(\mathbf{x})\| \leq \nu$ hold for all $\mathbf{x} \in \mathbb{R}^d$. In this subsection, we discuss more on this strong requirement “entire $\mathbb{R}^d$.”

In fact, the entire- $\mathbb{R}^d$ requirement is posed in our assumptions to simplify the exposition of the main results, and is not strictly necessary. One can easily check from our proofs that the assumptions are only applied to the beginning iterates $\mathbf{x}_{k,0}$ (for minibatch RR) or $\mathbf{y}_{k,0}$ (for local RR) of epochs. Hence, if these iterates lie in a bounded set, then the constants ν , τ , ρ , and λ may become much smaller, depending on problem instances. Actually, if we explicitly assume that the iterates lie in a compact set $\mathcal{S}$,⁵ then Assumptions 5.2–5.4 are even *guaranteed* to hold for some constants; e.g., for Assumption 5.2, we can choose

$$\nu := \max_{m \in [M], i \in [N], \mathbf{x} \in \mathcal{S}} \|\nabla f_i^m(\mathbf{x}) - \nabla F^m(\mathbf{x})\|,$$

since the maximum always exists.

However, assuming that the iterates lie in a specific set $\mathcal{S}$ can be problematic because the distance that the iterates travel depend on the objective functions. One cannot know a priori if all iterates will stay in a fixed set $\mathcal{S}$; hence, explicitly assuming bounded iterates should be avoided.

Then, a natural question is whether we can *prove* bounded iterates under some reasonable conditions, instead of assuming it. We point out that this can be done by applying the technique developed in Ahn et al. (2020, Theorem 1) to our upper bound theorems. Using the technique, a modified version of our Theorem 5.6 can be written as follows:

Theorem B.3 (Best-iterate version of Theorem 5.6). *Suppose that minibatch RR has parameters satisfying Assumption 5.1. Assume that all local component functions f_i^m are L -smooth, the global objective function F is μ -PL, and the set of global minima of F is nonempty and compact. Consider running the algorithm using step-size $\eta = \frac{B \log(MNK^2)}{\mu NK}$ and initialization $\mathbf{x}_{1,0} := \mathbf{x}_0$, for epochs $K \geq 6\kappa \log(MNK^2)$. Then, with probability at least $1 - \delta$,*

$$\min_{k \in [K+1]} F(\mathbf{x}_{k,0}) - F^* \leq \frac{F(\mathbf{x}_0) - F^*}{MNK^2} + \tilde{\mathcal{O}}\left(\frac{L^2}{\mu^3} \frac{\nu^2}{MNK^2}\right),$$

where the constant $\nu < \infty$ is defined as

$$\nu := \sup_{\mathbf{x}: F(\mathbf{x}) \leq F(\mathbf{x}_0)} \max_{i \in [N]} \max_{m \in [M]} \|\nabla f_i^m(\mathbf{x}) - \nabla F^m(\mathbf{x})\|. \quad (\text{B.113})$$

In the theorem, we used $\mathbf{x}_{K+1,0}$ to denote the last iterate of the algorithm $\mathbf{x}_{K, \frac{N}{B}}$. The-

⁵This *bounded iterates assumption* is indeed used in some existing results such as Haochen and Sra (2019b); Nagaraj et al. (2019); Rajput et al. (2020); Ahn et al. (2020).

orem B.3 differs from Theorem 5.6 in three aspects: 1) it considers the *best-iterate*, not the *last-iterate*; 2) it additionally assumes that the set of global minima of F is nonempty and compact, which always holds if F is strongly convex; and 3) it does not rely on Assumption 5.2, but instead “proves” it for the $F(\mathbf{x}_0)$ -sublevel set of F (B.113). Note that the constant ν (B.113) can be much smaller than the uniform bound required to make Assumption 5.2 hold for the entire $\mathbb{R}^d$. For Theorems 5.7, 5.13, and 5.14, we can also apply similar techniques to prove best-iterate bounds with smaller intra- and inter-machine deviation constants ν , τ , ρ , and λ ; we omit the precise statements.

We conclude this subsection with the proof of Theorem B.3.

Proof. The proof follows that of Ahn et al. (2020, Theorem 1).

Existence of ν . We first show the existence of $\nu < \infty$ (B.113). The global objective function F is μ -PL. If we denote the set of global minima of F as $\mathcal{X}^*$, the set $\mathcal{X}^*$ is nonempty and compact by assumption. Then, by Karimi et al. (2016, Theorem 2) μ -PL functions satisfy quadratic growth, i.e., denoting by $\mathbf{x}^*$ the closest global minimum in $\mathcal{X}^*$ to the point $\mathbf{x}$,

$$F(\mathbf{x}) - F^* \geq 2\mu \|\mathbf{x} - \mathbf{x}^*\|^2.$$

Define the sublevel set $\mathcal{S} := \{\mathbf{x} \mid F(\mathbf{x}) \leq F(\mathbf{x}_0)\}$. Due to the quadratic growth property, we have $F(\mathbf{x}_0) - F^* \geq F(\mathbf{x}) - F^* \geq 2\mu \|\mathbf{x} - \mathbf{x}^*\|^2$ for all $\mathbf{x} \in \mathcal{S}$. This implies that

$$\mathcal{S} := \{\mathbf{x} \in \mathbb{R}^d \mid F(\mathbf{x}) \leq F(\mathbf{x}_0)\} \subset \left\{ \mathbf{x} \in \mathbb{R}^d \mid \|\mathbf{x} - \mathbf{x}^*\|^2 \leq \frac{F(\mathbf{x}_0) - F^*}{2\mu} \right\}.$$

Since we assumed that $\mathcal{X}^*$ is compact, $\mathcal{S}$ is also bounded, and hence compact. Now, for any $m \in [M]$ and $i \in [N]$, $\|\nabla f_i^m(\mathbf{x}) - \nabla F^m(\mathbf{x})\|$ is a continuous function on a compact set $\mathcal{S}$, so there must exist a constant $\nu_i^m < \infty$ such that

$\|\nabla f_i^m(\mathbf{x}) - \nabla F^m(\mathbf{x})\| \leq \nu_i^m$ for all $\mathbf{x} \in \mathcal{S}$. Taking the maximum of ν_i^m over all m and i gives ν .

Proving the best-iterate bound. With the constant ν (B.113), if all the iterates $\{\mathbf{x}_{k,0}\}_{k \in [K+1]}$ stay within the sublevel set $\mathcal{S} := \{\mathbf{x} \mid F(\mathbf{x}) \leq F(\mathbf{x}_0)\}$, one can consider Assumption 5.2 to be true with constant ν . From this observation, we consider two cases:

1. All the iterates $\{\mathbf{x}_{k,0}\}_{k \in [K+1]}$ stay in the sublevel set $\mathcal{S}$.
2. There exists an iterate $\mathbf{x}_{k,0} \notin \mathcal{S}$.

In fact, the first case can be proven by exactly the same steps as Theorem 5.6, described in Appendix B.4.2.

For the second case, suppose that there exists an iterate $\mathbf{x}_{k,0}$ that escapes the sublevel set $\mathcal{S}$. Let $k' \in \{2, \dots, K+1\}$ be the first such k . Then, since $\mathbf{x}_{k'-1,0}$ is still in $\mathcal{S}$, it follows from (B.39) in Appendix B.4.2 that we have

$$F(\mathbf{x}_{k',0}) - F^* \leq (1 - \eta\mu N)(F(\mathbf{x}_{k'-1,0}) - F^*) + \frac{15\eta^3 L^2 \nu^2 (N^{3/2} - B^{3/2})^2}{MN} \log \frac{2NK}{B\delta}. \quad (\text{B.114})$$

However, the fact that $\mathbf{x}_{k',0} \notin \mathcal{S}$ and $\mathbf{x}_{k'-1,0} \in \mathcal{S}$ implies

$$F(\mathbf{x}_{k',0}) > F(\mathbf{x}_0) \geq F(\mathbf{x}_{k'-1,0}). \quad (\text{B.115})$$

Combining the two bounds (B.114) and (B.115), we get

$$0 < -\eta\mu N(F(\mathbf{x}_{k'-1,0}) - F^*) + \frac{15\eta^3 L^2 \nu^2 (N^{3/2} - B^{3/2})^2}{MN} \log \frac{2NK}{B\delta},$$

which implies

$$\min_{k \in [K+1]} F(\mathbf{x}_{k,0}) - F^* \leq F(\mathbf{x}_{k'-1,0}) - F^* < \frac{15\eta^2 L^2 \nu^2 (N^{3/2} - B^{3/2})^2}{\mu MN^2} \log \frac{2NK}{B\delta}.$$

Substituting $\eta = \frac{\log(MNK^2)}{\mu NK}^6$ gives the desired bound and finishes the proof. $\square$

B.5 Proof of lower bound for minibatch RR(Theorem 5.8)

For Theorem 5.8, we consider three step-size ranges and do case analysis for each of them. We construct functions for each corresponding step-size regime such that the convergence of minibatch RR is “slow” for the functions on their corresponding step-size regime. The final lower bound is the minimum among the lower bounds obtained for the three regimes. More concretely, we will construct three one-dimensional functions $F_1(x)$, $F_2(x)$, and $F_3(x)$ satisfying L -smoothness (5.1), μ -PL condition (5.2), and Assumption 5.2 such that⁷

⁶Recall that this is different from $\eta = \frac{B \log(MNK^2)}{\mu NK}$ in the theorem statement, because for the proofs, we consider an equivalent “rescaled” version of minibatch RR defined in the beginning of Appendix B.4.2.

⁷In fact, the functions constructed in this theorem are μ -strongly convex, which is stronger than μ -PL required in Definition 5.5.

- Minibatch RR on $F_1(x)$ with $\eta \leq \frac{B}{\mu NK}$ and initialization $x_0 = \frac{\nu}{\mu}$ results in

$$\mathbb{E}[F_1(x_{K, \frac{N}{B}})] = \Omega\left(\frac{\nu^2}{\mu}\right).$$

- Minibatch RR on $F_2(x)$ with $\eta \geq \frac{B}{\mu NK}$ and $\eta \leq \frac{B}{513LN}$ and initialization $x_0 = 0$ results in

$$\mathbb{E}[F_2(x_{K, \frac{N}{B}})] = \Omega\left(\frac{\nu^2}{\mu MNK^2}\right).$$

Note that the step-size range requires $K \geq 513\kappa$, hence this lower bound occurs only in the “large-epoch” regime, i.e., $K \gtrsim \kappa$.

- Minibatch RR on $F_3(x)$ with $\eta \geq \frac{B}{\mu NK}$ and $\eta \geq \frac{B}{513LN}$ and initialization $x_0 = 0$ results in

$$\mathbb{E}[F_3(x_{K, \frac{N}{B}})] = \Omega\left(\frac{\nu^2}{\mu MNK}\right).$$

Then, the three dimensional function $F([x, y, z]^\top) = F_1(x) + F_2(y) + F_3(z)$ will show bad convergence in any step-size regime. Furthermore,

$$\mu \mathbf{I} \preceq \min(\nabla^2 F_1, \nabla^2 F_2, \nabla^2 F_3) \mathbf{I} \preceq \nabla^2 F \preceq \max(\nabla^2 F_1, \nabla^2 F_2, \nabla^2 F_3) \mathbf{I} \preceq L \mathbf{I},$$

that is, if F_1 , F_2 and F_3 are μ -strongly convex and L -smooth, then so is F . Moreover, since the component functions in each coordinate are designed to satisfy Assumption 5.2 with ν , the resulting three dimensional function F also satisfies Assumption 5.2 with $\sqrt{3}\nu$.

Since the final lower bound is the minimum among the lower bounds obtained in the step-size ranges, the lower bound becomes $\Omega\left(\frac{\nu^2}{\mu MNK^2}\right)$ if $K \geq 513\kappa$, and $\Omega\left(\frac{\nu^2}{\mu MNK}\right)$ if $K < 513\kappa$ (in which case the second step-size range does not exist).

In the subsequent subsections, we prove the lower bounds for F_1 , F_2 , and F_3 separately.

B.5.1 Lower bound for $\eta \leq \frac{B}{\mu NK}$

Consider the case where every function at every machine is the same: for all $i \in [N]$ and $m \in [M]$, $f_i^m(x) := \frac{\mu x^2}{2}$. Hence, $F_1(x) = \frac{\mu x^2}{2}$.

Let $x_{k,0}$ and $x_{k, \frac{N}{B}}$ denote the iterates where the k -th epoch starts and ends respectively.

Then,

$$x_{k+1,0} = x_{k,\frac{N}{B}} = (1 - \eta\mu)^{\frac{N}{B}} x_{k,0}.$$

Initializing at $x_{1,0} = \frac{\nu}{\mu}$ and unrolling this for K epochs, we get

$$x_{K,\frac{N}{B}} = (1 - \eta\mu)^{\frac{NK}{B}} \cdot \frac{\nu}{\mu} \geq \left(1 - \frac{B}{NK}\right)^{\frac{NK}{B}} \cdot \frac{\nu}{\mu} \geq \frac{\nu}{4\mu},$$

since $N \geq 2$, $K \geq 1$, and B divides N . Hence, $F_1(x_{K,\frac{N}{B}}) = \Omega(\frac{\nu^2}{\mu})$.

B.5.2 Lower bound for $\eta \geq \frac{B}{\mu NK}$ and $\eta \leq \frac{B}{513LN}$

For most part of this subsection, we consider iterates within a single epoch, and hence we will omit the subscripts denoting epochs. Let x_0 denote the iterate at the beginning of the epoch, and x_i denote the iterate after the i -th communication round in that epoch. In our construction, each machine will have the same set of component functions, that is, there will be no inter-machine deviation. We therefore omit the superscript m from the local component functions f_i^m . The function we construct for the lower bound and its component functions are as follows:

$$\begin{aligned} F_2(x) &:= \frac{1}{N} \left(\sum_{i=1}^{\frac{N}{2}} f_{+1}(x) + \sum_{i=\frac{N}{2}+1}^N f_{-1}(x) \right), \text{ where} \\ f_{+1}(x) &:= (L1_{x \leq 0} + \mu 1_{x > 0}) \frac{x^2}{2} + \nu x, \text{ and} \\ f_{-1}(x) &:= (L1_{x \leq 0} + \mu 1_{x > 0}) \frac{x^2}{2} - \nu x \end{aligned}$$

Note that the function $F_2(x) = (L1_{x \leq 0} + \mu 1_{x > 0}) \frac{x^2}{2}$ is μ -strongly convex and L -smooth with minimizer at 0, and also satisfies Assumption 5.2.

Let σ^m be a random permutation of $\frac{N}{2} + 1$'s and $\frac{N}{2} - 1$'s. Then, machine m computes gradients on f_{-1} and f_{+1} in the order given by σ^m . Let σ_j^m denote the j -th ordered element of σ^m . Then,

$$\nabla f_{\sigma_j^m}(x) = (L1_{x \leq 0} + \mu 1_{x > 0})x + \nu \sigma_j^m.$$

Hence, the last iterate of an epoch, $x_{\frac{N}{B}}$, is given by

$$\begin{aligned}
x_{\frac{N}{B}} - x_0 &= \sum_{i=0}^{\frac{N}{B}-1} \left(-\frac{\eta}{MB} \sum_{m=1}^M \sum_{j=iB+1}^{(i+1)B} \nabla f_{\sigma_j^m}(x_i) \right) \\
&= \sum_{i=0}^{\frac{N}{B}-1} \left(-\frac{\eta}{MB} \sum_{m=1}^M \sum_{j=iB+1}^{(i+1)B} ((L1_{x_i \leq 0} + \mu 1_{x_i > 0})x_i + \nu \sigma_j^m) \right) \\
&= \sum_{i=0}^{\frac{N}{B}-1} \left(-\frac{\eta}{MB} \sum_{m=1}^M \sum_{j=iB+1}^{(i+1)B} (L1_{x_i \leq 0} + \mu 1_{x_i > 0})x_i \right) \quad (\text{Since } \sum_{j=1}^N \sigma_j^m = 0) \\
&= -\eta \sum_{i=0}^{\frac{N}{B}-1} (L1_{x_i \leq 0} + \mu 1_{x_i > 0})x_i. \tag{B.116}
\end{aligned}$$

Thus, $\mathbb{E}[x_{\frac{N}{B}} - x_0] = -\eta \sum_{i=0}^{\frac{N}{B}-1} \mathbb{E}[(L1_{x_i \leq 0} + \mu 1_{x_i > 0})x_i]$. We want to prove that $\mathbb{E}[x_{\frac{N}{B}}]$ keeps increasing over an epoch, that is $\mathbb{E}[x_{\frac{N}{B}} - x_0] > 0$ when x_0 is close enough to the minimizer 0.

For this, we first consider the case where the first iterate x_0 of the epoch satisfies $x_0 \geq 0$. The $x_0 < 0$ case will be considered later. For the case $x_0 \geq 0$, we will show that whenever x_0 is small, the expected amount of update made in the $(i+1)$ -th iteration, $\mathbb{E}[(L1_{x_i \leq 0} + \mu 1_{x_i > 0})x_i]$, is negative in the first half of the epoch and not too big in the second half.

We use the following lemmas, proven in Appendices B.6.1 and B.6.2, respectively.

Lemma B.4. For $x_0 \geq 0$, $0 \leq i \leq \lfloor \frac{N}{2B} \rfloor$, $\eta \leq \frac{B}{513LN}$, and $\frac{L}{\mu} \geq 7695$,

$$\mathbb{E}[(L1_{x_i \leq 0} + \mu 1_{x_i > 0})x_i] \leq \frac{6}{7}Lx_0 - \frac{\eta L \nu}{1536} \sqrt{\frac{i}{MB}}.$$

Lemma B.5. For $x_0 \geq 0$, $0 \leq i \leq \frac{N}{B} - 1$, and $\eta \leq \frac{B}{513LN}$,

$$\mathbb{E}[(L1_{x_i \leq 0} + \mu 1_{x_i > 0})x_i] \leq \mu \left(1 + \frac{513i\eta L}{512} \right) x_0 + \frac{513\eta \mu \nu}{512} \sqrt{\frac{i}{MB}}.$$

The key intuition is that, for $\frac{L}{\mu}$ big enough, we can use the lemmas above in (B.116) to get

$$\begin{aligned}
\mathbb{E}[x_{\frac{N}{B}} - x_0] &= -\eta \sum_{i=0}^{\frac{N}{B}-1} (L1_{x_i \leq 0} + \mu 1_{x_i > 0})x_i \\
&\approx \Omega \left(\eta \frac{N}{B} \eta L \nu \sqrt{\frac{N/B}{MB}} \right),
\end{aligned}$$

whenever $|x_0|$ is small. Multiplying the above by K (for K epochs) will give us the required lower bound (up to factors of $\frac{L}{\mu}$). We will make this approximate calculation precise in the rest of the proof.

Using the two lemmas above in (B.116), we get that

$$\begin{aligned}
& \mathbb{E}[x_{\frac{N}{B}} - x_0] \\
&= -\eta \sum_{i=0}^{\frac{N}{B}-1} \mathbb{E}[(L1_{x_i \leq 0} + \mu 1_{x_i > 0})x_i] \\
&= -\eta \sum_{i=0}^{\lfloor \frac{N}{2B} \rfloor} \mathbb{E}[(L1_{x_i \leq 0} + \mu 1_{x_i > 0})x_i] - \eta \sum_{i=\lfloor \frac{N}{2B} \rfloor+1}^{\frac{N}{B}-1} \mathbb{E}[(L1_{x_i \leq 0} + \mu 1_{x_i > 0})x_i] \\
&\geq -\eta \sum_{i=0}^{\lfloor \frac{N}{2B} \rfloor} \left(\frac{6}{7}Lx_0 - \frac{\eta L\nu}{1536} \sqrt{\frac{i}{MB}} \right) \\
&\quad - \eta \sum_{i=\lfloor \frac{N}{2B} \rfloor+1}^{\frac{N}{B}-1} \left(\mu \left(1 + \frac{513i\eta L}{512} \right) x_0 + \frac{513\eta\mu\nu}{512} \sqrt{\frac{i}{MB}} \right). \tag{B.117}
\end{aligned}$$

Since $i\eta L \leq \frac{\eta LN}{B} \leq \frac{1}{513}$, $\mu \leq \frac{L}{7695}$, and $N/B \geq 2$, the following bound holds:

$$\begin{aligned}
& \sum_{i=0}^{\lfloor \frac{N}{2B} \rfloor} \frac{6}{7}L + \sum_{i=\lfloor \frac{N}{2B} \rfloor+1}^{\frac{N}{B}-1} \mu \left(1 + \frac{513i\eta L}{512} \right) \\
&\leq \left(\left\lfloor \frac{N}{2B} \right\rfloor + 1 \right) \frac{6L}{7} + \left(\frac{N}{B} - \left\lfloor \frac{N}{2B} \right\rfloor - 1 \right) \frac{L}{7695} \left(1 + \frac{1}{512} \right) \\
&\leq \frac{6LN}{7B} + \frac{LN}{7680B} \leq \frac{7LN}{8B}. \tag{B.118}
\end{aligned}$$

Also, note that $\lfloor \frac{N}{2B} \rfloor \geq \frac{N}{3B}$ whenever $N/B \geq 2$. We have

$$\sum_{i=0}^{\lfloor \frac{N}{2B} \rfloor} \sqrt{i} \geq \int_0^{\lfloor \frac{N}{2B} \rfloor} \sqrt{t} dt = \frac{2}{3} \left(\left\lfloor \frac{N}{2B} \right\rfloor \right)^{3/2} \geq \frac{2}{3} \left(\frac{N}{3B} \right)^{3/2} = \frac{2N^{3/2}}{9\sqrt{3}B^{3/2}}, \tag{B.119}$$

$$\begin{aligned}
\sum_{i=\lfloor \frac{N}{2B} \rfloor+1}^{\frac{N}{B}-1} \sqrt{i} &\leq \int_{\lfloor \frac{N}{2B} \rfloor+1}^{\frac{N}{B}} \sqrt{t} dt \leq \frac{2}{3} \left[\left(\frac{N}{B} \right)^{3/2} - \left(\left\lfloor \frac{N}{2B} \right\rfloor + 1 \right)^{3/2} \right] \\
&\leq \frac{2}{3} \left[\left(\frac{N}{B} \right)^{3/2} - \left(\frac{N}{2B} \right)^{3/2} \right] = \frac{(2\sqrt{2}-1)N^{3/2}}{3\sqrt{2}B^{3/2}} \leq \frac{N^{3/2}}{2B^{3/2}}. \tag{B.120}
\end{aligned}$$

Substituting the bounds (B.118), (B.119), and (B.120) into (B.117), and using $\mu \leq \frac{L}{7695}$,

$$\begin{aligned}\mathbb{E}[x_{\frac{N}{B}} - x_0] &\geq -\frac{7\eta LN}{8B}x_0 + \frac{\eta^2 L\nu N^{3/2}}{6912\sqrt{3}M^{1/2}B^2} - \frac{513\eta^2\mu\nu N^{3/2}}{1024M^{1/2}B^2} \\ &\geq -\frac{7\eta LN}{8B}x_0 + \frac{\eta^2 L\nu N^{3/2}}{56000M^{1/2}B^2}.\end{aligned}\tag{B.121}$$

For the other case $x_0 < 0$, we have the following Lemma, which we prove in Appendix B.6.3:

Lemma B.6. *If $\eta \leq \frac{B}{513NL}$ and an epoch starts at $x_0 < 0$, then*

$$\mathbb{E}[x_{\frac{N}{B}} \mid x_0 < 0] \geq \left(1 - \frac{7\eta LN}{8B}\right)x_0.$$

Further, if the first epoch of the algorithm is initialized at 0, then for any starting iterate x_0 of any following epoch, we have $\mathbb{P}(x_0 \geq 0) \geq 1/2$.

Using (B.121) and Lemma B.6 we get

$$\begin{aligned}\mathbb{E}[x_{\frac{N}{B}}] &= \mathbb{P}(x_0 \geq 0)\mathbb{E}[x_{\frac{N}{B}} \mid x_0 \geq 0] + \mathbb{P}(x_0 < 0)\mathbb{E}[x_{\frac{N}{B}} \mid x_0 < 0] \\ &\geq \mathbb{P}(x_0 \geq 0) \left(\left(1 - \frac{7\eta LN}{8B}\right)x_0 + \frac{\eta^2 L\nu N^{3/2}}{56000M^{1/2}B^2} \right) + \mathbb{P}(x_0 < 0) \left(1 - \frac{7\eta LN}{8B}\right)x_0 \\ &\geq \left(1 - \frac{7\eta LN}{8B}\right)x_0 + \frac{\eta^2 L\nu N^{3/2}}{112000M^{1/2}B^2}.\end{aligned}$$

Thus far, we have characterized the expected per-epoch update, starting from the initial iterate x_0 and iterating until the last iterate $x_{\frac{N}{B}}$ of the epoch. Now recall that we run the algorithm for K epochs. Using $x_{k,i}$ to denote the i -th iterate of the k -th epoch, we get a lower bound on the expectation of the last iterate $x_{k,\frac{N}{B}}$ if we initialize at $x_{1,0} = 0$:

$$\begin{aligned}\mathbb{E}[x_{K,\frac{N}{B}}] &\geq \left(1 - \frac{7\eta LN}{8B}\right)^K x_{1,0} + \frac{\eta^2 L\nu N^{3/2}}{112000M^{1/2}B^2} \sum_{k=0}^{K-1} \left(1 - \frac{7\eta LN}{8B}\right)^k \\ &= \frac{\eta^2 L\nu N^{3/2}}{112000M^{1/2}B^2} \frac{1 - \left(1 - \frac{7\eta LN}{8B}\right)^K}{\frac{7\eta LN}{8B}} \\ &= \frac{\eta\nu N^{1/2}}{98000M^{1/2}B} \left(1 - \left(1 - \frac{7\eta LN}{8B}\right)^K\right) \\ &\geq \frac{\eta\nu N^{1/2}}{98000M^{1/2}B} \left(1 - \left(1 - \frac{7L}{8\mu K}\right)^K\right).\end{aligned}\tag{Since $\eta \geq \frac{B}{\mu NK}$ }$$

Note that since $\frac{L}{\mu} \geq 7695$ and $K \geq \frac{513L}{\mu}$ (which is implied by $\frac{B}{\mu NK} \leq \eta \leq \frac{B}{513LN}$),

$$1 - \left(1 - \frac{7L}{8\mu K}\right)^K \geq 1 - e^{-\frac{7L}{8\mu}} \geq 1 - e^{-6733} \approx 1.$$

Hence, we get from $\eta \geq \frac{B}{\mu NK}$ that

$$\mathbb{E}[x_{K, \frac{N}{B}}] = \Omega\left(\frac{\eta \nu N^{1/2}}{M^{1/2} B}\right) = \Omega\left(\frac{\nu}{\mu M^{1/2} N^{1/2} K}\right),$$

and by Jensen's inequality, we finally have

$$\mathbb{E}[F(x_{K, \frac{N}{B}})] \geq \frac{1}{2} \mathbb{E}[\mu x_{K, \frac{N}{B}}^2] = \Omega(\mu \mathbb{E}[x_{K, \frac{N}{B}}]^2) = \Omega\left(\frac{\nu^2}{\mu M N K^2}\right).$$

B.5.3 Lower bound for $\eta \geq \frac{B}{\mu NK}$ and $\eta \geq \frac{B}{513LN}$

Similar to earlier parts of the proof, here as well, each machine will have the same component functions, that is, there will be no inter-machine deviation. The proof uses a similar construction as Safran and Shamir (2020, 2021):

$$F_3(x) := \frac{1}{N} \left(\sum_{i=1}^{\frac{N}{2}} f_{+1}(x) + \sum_{i=\frac{N}{2}+1}^N f_{-1}(x) \right), \text{ where}$$

$$f_{+1}(x) := \frac{Lx^2}{2} + \nu x, \text{ and } f_{-1}(x) := \frac{Lx^2}{2} - \nu x.$$

Hence, $F_3(x) = \frac{Lx^2}{2}$, and has its minimizer at 0.

We first compute the expected “progress” over a given epoch. For simplicity, let us omit the subscript for epochs for now. Let x_0 denote the iterate at the beginning of the epoch and x_i denote the iterate after the i -th communication round in that epoch. For a given epoch, let σ^m be the permutation of $\frac{N}{2} + 1$'s and $\frac{N}{2} - 1$'s, sampled by machine m . Then,

$$\begin{aligned} x_{\frac{N}{B}} &= x_{\frac{N}{B}-1} - \frac{\eta}{MB} \sum_{m=1}^M \sum_{j=(\frac{N}{B}-1)B+1}^N (Lx_{\frac{N}{B}-1} + \nu \sigma_j^m) \\ &= (1 - \eta L)x_{\frac{N}{B}-1} - \frac{\eta \nu}{MB} \sum_{m=1}^M \sum_{j=(\frac{N}{B}-1)B+1}^N \sigma_j^m \\ &= \dots \end{aligned}$$

$$= (1 - \eta L)^{\frac{N}{B}} x_0 - \frac{\eta \nu}{MB} \sum_{i=1}^{\frac{N}{B}} (1 - \eta L)^{\frac{N}{B}-i} \sum_{m=1}^M \sum_{j=(i-1)B+1}^{iB} \sigma_j^m.$$

For the rest of this proof, x_i^2 refers to the square of the i -th iterate. Then,

$$\begin{aligned} \mathbb{E}[x_{\frac{N}{B}}^2] &= (1 - \eta L)^{\frac{2N}{B}} x_0^2 - \frac{2\eta \nu (1 - \eta L)^{\frac{N}{B}} x_0}{MB} \mathbb{E} \left[\left(\sum_{i=1}^{\frac{N}{B}} (1 - \eta L)^{\frac{N}{B}-i} \sum_{m=1}^M \sum_{j=(i-1)B+1}^{iB} \sigma_j^m \right) \right] \\ &\quad + \frac{\eta^2 \nu^2}{M^2 B^2} \mathbb{E} \left[\left(\sum_{i=1}^{\frac{N}{B}} (1 - \eta L)^{\frac{N}{B}-i} \sum_{m=1}^M \sum_{j=(i-1)B+1}^{iB} \sigma_j^m \right)^2 \right] \\ &= (1 - \eta L)^{\frac{2N}{B}} x_0^2 + \frac{\eta^2 \nu^2}{M^2 B^2} \mathbb{E} \left[\left(\sum_{i=1}^{\frac{N}{B}} (1 - \eta L)^{\frac{N}{B}-i} \sum_{m=1}^M \sum_{j=(i-1)B+1}^{iB} \sigma_j^m \right)^2 \right], \end{aligned} \quad (\text{B.122})$$

where we used the fact that $\mathbb{E}[\sigma_j^m] = 0$. Further, because σ^m and $\sigma^{m'}$ are independent and identically distributed for different m and m' , we get that

$$\begin{aligned} &\mathbb{E} \left[\left(\sum_{i=1}^{\frac{N}{B}} (1 - \eta L)^{\frac{N}{B}-i} \sum_{m=1}^M \sum_{j=(i-1)B+1}^{iB} \sigma_j^m \right)^2 \right] \\ &= \mathbb{E} \left[\left(\sum_{m=1}^M \sum_{i=1}^{\frac{N}{B}} (1 - \eta L)^{\frac{N}{B}-i} \sum_{j=(i-1)B+1}^{iB} \sigma_j^m \right)^2 \right] \\ &= \sum_{m=1}^M \mathbb{E} \left[\left(\sum_{i=1}^{\frac{N}{B}} (1 - \eta L)^{\frac{N}{B}-i} \sum_{j=(i-1)B+1}^{iB} \sigma_j^m \right)^2 \right] \\ &\quad + \sum_{m \neq m'} \mathbb{E} \left[\sum_{i=1}^{\frac{N}{B}} (1 - \eta L)^{\frac{N}{B}-i} \sum_{j=(i-1)B+1}^{iB} \sigma_j^m \right] \mathbb{E} \left[\sum_{i=1}^{\frac{N}{B}} (1 - \eta L)^{\frac{N}{B}-i} \sum_{j=(i-1)B+1}^{iB} \sigma_j^{m'} \right] \\ &= M \mathbb{E} \left[\left(\sum_{i=1}^{\frac{N}{B}} (1 - \eta L)^{\frac{N}{B}-i} \sum_{j=(i-1)B+1}^{iB} \sigma_j^1 \right)^2 \right], \end{aligned}$$

where the last equality used the fact that $\mathbb{E}[\sigma_j^m] = 0$ for all $m \in [M]$ and $i \in [N]$, and that σ^m are identically distributed. Since we only consider the permutation σ^1 (i.e., the one for machine 1) from now on, we henceforth omit the superscript. Substituting this to (B.122)

gives

$$\mathbb{E}[x_{\frac{N}{B}}^2] = (1 - \eta L)^{\frac{2N}{B}} x_0^2 + \underbrace{\frac{\eta^2 \nu^2}{MB^2} \mathbb{E} \left[\left(\sum_{i=1}^{\frac{N}{B}} (1 - \eta L)^{\frac{N}{B}-i} \sum_{j=(i-1)B+1}^{iB} \sigma_j \right)^2 \right]}_{=:\Phi}. \quad (\text{B.123})$$

From (B.123), we have calculated the per-epoch expected update. Recall that we run the algorithm for K epochs. Using $x_{k,i}$ to denote the i -th iterate of the k -th epoch, we get a lower bound on the expectation of the last iterate $x_{k,\frac{N}{B}}$ squared:

$$\mathbb{E}[x_{k,\frac{N}{B}}^2] = (1 - \eta L)^{\frac{2NK}{B}} x_{1,0}^2 + \frac{\eta^2 \nu^2}{MB^2} \Phi \sum_{k=0}^{K-1} (1 - \eta L)^{\frac{2Nk}{B}} \geq \frac{\eta^2 \nu^2}{MB^2} \Phi, \quad (\text{B.124})$$

where the inequality used $x_{1,0} = 0$ and $\sum_{k=0}^{K-1} (1 - \eta L)^{\frac{2Nk}{B}} \geq (1 - \eta L)^0 = 1$. Next, we analyze the expectation term, i.e., Φ , defined in (B.123).

$$\begin{aligned} \Phi &:= \mathbb{E} \left[\left(\sum_{i=1}^{\frac{N}{B}} (1 - \eta L)^{\frac{N}{B}-i} \sum_{j=(i-1)B+1}^{iB} \sigma_j \right)^2 \right] \\ &= \sum_{j=1}^N (1 - \eta L)^{2(\frac{N}{B}-\lfloor \frac{j-1}{B} \rfloor - 1)} \sigma_j^2 + \sum_{j \neq j'} (1 - \eta L)^{\frac{N}{B}-\lfloor \frac{j-1}{B} \rfloor - 1} (1 - \eta L)^{\frac{N}{B}-\lfloor \frac{j'-1}{B} \rfloor - 1} \mathbb{E}[\sigma_j \sigma_{j'}] \end{aligned}$$

Noting that $\sigma_j^2 = 1$ and $\mathbb{E}[\sigma_j \sigma_{j'}] = -\frac{1}{N-1}$, we get

$$\begin{aligned} \Phi &= B \sum_{j=0}^{\frac{N}{B}-1} (1 - \eta L)^{2j} - \frac{1}{N-1} \sum_{j \neq j'} (1 - \eta L)^{\frac{N}{B}-\lfloor \frac{j-1}{B} \rfloor - 1} (1 - \eta L)^{\frac{N}{B}-\lfloor \frac{j'-1}{B} \rfloor - 1} \\ &= B \sum_{j=0}^{\frac{N}{B}-1} (1 - \eta L)^{2j} - \frac{1}{N-1} \left(\left(\sum_{j=1}^N (1 - \eta L)^{\frac{N}{B}-\lfloor \frac{j-1}{B} \rfloor - 1} \right)^2 \right. \\ &\quad \left. - \sum_{j=1}^N (1 - \eta L)^{2(\frac{N}{B}-\lfloor \frac{j-1}{B} \rfloor - 1)} \right) \\ &= B \sum_{j=0}^{\frac{N}{B}-1} (1 - \eta L)^{2j} - \frac{1}{N-1} \left(B^2 \left(\sum_{j=0}^{\frac{N}{B}-1} (1 - \eta L)^j \right)^2 - B \sum_{j=0}^{\frac{N}{B}-1} (1 - \eta L)^{2j} \right) \\ &= \frac{B^2}{N-1} \left(\frac{N}{B} \sum_{j=0}^{\frac{N}{B}-1} (1 - \eta L)^{2j} - \left(\sum_{j=0}^{\frac{N}{B}-1} (1 - \eta L)^j \right)^2 \right) \end{aligned}$$

$$= \frac{B^2(\frac{N}{B}-1)}{N-1} \left(\left(1 + \frac{1}{\frac{N}{B}-1} \right) \sum_{j=0}^{\frac{N}{B}-1} (1-\eta L)^{2j} - \frac{1}{\frac{N}{B}-1} \left(\sum_{j=0}^{\frac{N}{B}-1} (1-\eta L)^j \right)^2 \right). \quad (\text{B.125})$$

Note that the term in the parenthesis is exactly the right hand side of Equation (23) in Safran and Shamir (2020) modulo n and α replaced with $\frac{N}{B}$ and ηL , respectively. Hence, by Lemma 1 of Safran and Shamir (2020), we have

$$\left(1 + \frac{1}{\frac{N}{B}-1} \right) \sum_{j=0}^{\frac{N}{B}-1} (1-\eta L)^{2j} - \frac{1}{\frac{N}{B}-1} \left(\sum_{j=0}^{\frac{N}{B}-1} (1-\eta L)^j \right)^2 \geq c \cdot \min \left\{ \frac{1}{\eta L}, \frac{\eta^2 L^2 N^3}{B^3} \right\}, \quad (\text{B.126})$$

for some universal constant $c > 0$. Using the fact that $\eta \geq \frac{B}{513LN}$, it is easy to check that the RHS of (B.126) is lower-bounded by $\frac{c'}{\eta L}$, where $c' > 0$ is a universal constant. Combining (B.124), (B.125), and (B.126) gives

$$\mathbb{E}[x_{k, \frac{N}{B}}^2] \geq \frac{\eta^2 \nu^2}{MB^2} \cdot \frac{B^2(\frac{N}{B}-1)}{N-1} \cdot \frac{c'}{\eta L} = \frac{c' \eta \nu^2}{LMB} \frac{N-B}{N-1}.$$

Since $2B$ divides N , we have $B \leq N/2$. Since $N \geq 2$, we have $\frac{N-B}{N-1} \geq \frac{N}{2(N-1)} \geq \frac{1}{2}$. Using this and the fact that $\eta \geq \frac{B}{\mu NK}$, we get

$$\mathbb{E}[F_3(x_{k, \frac{N}{B}})] = \frac{L}{2} \mathbb{E}[x_{k, \frac{N}{B}}^2] \geq \frac{c' \nu^2}{4\mu MNK}.$$

B.6 Proofs of helper lemmas for Appendix B.5

B.6.1 Proof of Lemma B.4

First, if $i = 0$ then the lemma trivially holds, because $x_0 \geq 0$ gives

$$\mathbb{E}[(L1_{x_0 \leq 0} + \mu 1_{x_0 > 0})x_0] = \mu x_0 \leq \frac{6}{7} L x_0.$$

The inequality holds because $\frac{L}{\mu} \geq 7695$.

For the rest of the proof, we consider the case $1 \leq i \leq \frac{N}{2B}$. By the law of total expectation we have

$$\begin{aligned} & \mathbb{E}[(L1_{x_i \leq 0} + \mu 1_{x_i > 0})x_i] \\ &= \mathbb{P} \left(\sum_{m=1}^M \sum_{j=1}^{iB} \sigma_j^m > 0 \right) \mathbb{E} \left[(L1_{x_i \leq 0} + \mu 1_{x_i > 0})x_i \middle| \sum_{m=1}^M \sum_{j=1}^{iB} \sigma_j^m > 0 \right] \end{aligned}$$

$$\begin{aligned}
& + \mathbb{P} \left(\sum_{m=1}^M \sum_{j=1}^{iB} \sigma_j^m \leq 0 \right) \mathbb{E} \left[(L1_{x_i \leq 0} + \mu 1_{x_i > 0}) x_i \left| \sum_{m=1}^M \sum_{j=1}^{iB} \sigma_j^m \leq 0 \right. \right] \\
& \leq \mathbb{P} \left(\sum_{m=1}^M \sum_{j=1}^{iB} \sigma_j^m > 0 \right) L \mathbb{E} \left[x_i \left| \sum_{m=1}^M \sum_{j=1}^{iB} \sigma_j^m > 0 \right. \right] \\
& + \mathbb{P} \left(\sum_{m=1}^M \sum_{j=1}^{iB} \sigma_j^m \leq 0 \right) \mu \mathbb{E} \left[x_i \left| \sum_{m=1}^M \sum_{j=1}^{iB} \sigma_j^m \leq 0 \right. \right], \tag{B.127}
\end{aligned}$$

where the last inequality used the fact that $(L1_{t \leq 0} + \mu 1_{t > 0})t \leq Lt$ and $(L1_{t \leq 0} + \mu 1_{t > 0})t \leq \mu t$ for any $t \in \mathbb{R}$.

Define $\mathcal{E} := \sum_{m=1}^M \sum_{j=1}^{iB} \sigma_j^m$. We handle each of the two expectations in (B.127) separately. We first bound $\mathbb{E}[x_i | \mathcal{E} > 0]$.

$$\begin{aligned}
\mathbb{E}[x_i | \mathcal{E} > 0] &= \mathbb{E} \left[x_0 - \sum_{j=0}^{i-1} \frac{\eta}{MB} \sum_{m=1}^M \sum_{k=jB+1}^{(j+1)B} (\nu \sigma_k^m + (L1_{x_j \leq 0} + \mu 1_{x_j > 0}) x_j) \left| \mathcal{E} > 0 \right. \right] \\
&= \mathbb{E} \left[x_0 - \sum_{j=0}^{i-1} \frac{\eta}{MB} \sum_{m=1}^M \sum_{k=jB+1}^{(j+1)B} (\nu \sigma_k^m + (L1_{x_j \leq 0} + \mu 1_{x_j > 0})(x_j - x_0)) \left| \mathcal{E} > 0 \right. \right] \\
&\quad + \mathbb{E} \left[- \sum_{j=0}^{i-1} \frac{\eta}{MB} \sum_{m=1}^M \sum_{k=jB+1}^{(j+1)B} (L1_{x_j \leq 0} + \mu 1_{x_j > 0}) x_0 \left| \mathcal{E} > 0 \right. \right] \\
&= x_0 \mathbb{E} \left[1 - \eta \sum_{j=0}^{i-1} (L1_{x_j \leq 0} + \mu 1_{x_j > 0}) \left| \mathcal{E} > 0 \right. \right] - \frac{\eta \nu}{MB} \mathbb{E}[\mathcal{E} | \mathcal{E} > 0] \\
&\quad - \eta \sum_{j=0}^{i-1} \mathbb{E}[(L1_{x_j \leq 0} + \mu 1_{x_j > 0})(x_j - x_0) | \mathcal{E} > 0] \\
&\leq x_0 \mathbb{E} \left[1 - \eta \sum_{j=0}^{i-1} (L1_{x_j \leq 0} + \mu 1_{x_j > 0}) \left| \mathcal{E} > 0 \right. \right] - \frac{\eta \nu}{MB} \mathbb{E}[\mathcal{E} | \mathcal{E} > 0] \\
&\quad + \eta L \sum_{j=0}^{i-1} \mathbb{E}[|x_j - x_0| | \mathcal{E} > 0]. \tag{B.128}
\end{aligned}$$

Next, we use the following lemma to bound the conditional expectations that arise in (B.128). This lemma is proven in Appendix B.6.4 and it may be of independent interest to readers.

Lemma B.7. *For $m \in [M]$, let σ^m be a random permutation of $\frac{N}{2} + 1$'s and $\frac{N}{2} - 1$'s. Then, for any $i \leq \frac{N}{2}$ and $k \leq \frac{B}{2}$, we have*

$$\frac{1}{64} \left(\sqrt{\frac{i}{M}} + \sqrt{k} \right) \leq \mathbb{E} \left[\left| \left(\frac{1}{M} \sum_{m=1}^M \sum_{j=1}^i \sigma_j^m \right) + \sum_{j=i+1}^{i+k} \sigma_j^M \right| \right].$$

Furthermore, for any $0 \leq i \leq N$ and $0 \leq k \leq N$ satisfying $i + k \leq N$, we have

$$\mathbb{E} \left[\left| \left(\frac{1}{M} \sum_{m=1}^M \sum_{j=1}^i \sigma_j^m \right) + \sum_{j=i+1}^{i+k} \sigma_j^M \right| \right] \leq \sqrt{\frac{i}{M}} + \sqrt{k}.$$

Lastly, for any $0 \leq i \leq \frac{N}{2}$ and $0 \leq k \leq \frac{B}{2}$ satisfying $i + k \geq 1$, we have

$$\mathbb{P} \left(\sum_{m=1}^M \sum_{j=1}^i \sigma_j^m + M \sum_{j=i+1}^{i+k} \sigma_j^M > 0 \right) = \mathbb{P} \left(\sum_{m=1}^M \sum_{j=1}^i \sigma_j^m + M \sum_{j=i+1}^{i+k} \sigma_j^M < 0 \right) \geq \frac{1}{6}.$$

Lemma B.7 implies that $\frac{1}{M} \mathbb{E} [\mathcal{E} | \mathcal{E} > 0] \in \left[\frac{1}{64} \sqrt{\frac{iB}{M}}, \sqrt{\frac{iB}{M}} \right]$ and $\mathbb{P}(\mathcal{E} > 0) = \mathbb{P}(\mathcal{E} < 0) \geq 1/6$. From this, we get

$$\frac{\eta\nu}{MB} \mathbb{E} [\mathcal{E} | \mathcal{E} > 0] \geq \frac{\eta\nu}{64} \sqrt{\frac{i}{MB}}, \quad (\text{B.129})$$

$$\mathbb{E}[|x_j - x_0| | \mathcal{E} > 0] \leq \frac{\mathbb{E}[|x_j - x_0|]}{\mathbb{P}(\mathcal{E} > 0)} \leq 6 \mathbb{E}[|x_j - x_0|]. \quad (\text{B.130})$$

Also, since $\eta \leq \frac{B}{LN}$ we have

$$1 - i\eta\mu \geq 1 - \eta \sum_{j=0}^{i-1} (L1_{x_j \leq 0} + \mu 1_{x_j > 0}) \geq 1 - \frac{\eta LN}{B} \geq 0,$$

which implies that

$$x_0 \mathbb{E} \left[1 - \eta \sum_{j=0}^{i-1} (L1_{x_j \leq 0} + \mu 1_{x_j > 0}) \middle| \mathcal{E} > 0 \right] \leq (1 - i\eta\mu)x_0. \quad (\text{B.131})$$

Substituting (B.129), (B.130), and (B.131) to (B.128), we obtain

$$\mathbb{E} [x_i | \mathcal{E} > 0] \leq (1 - i\eta\mu)x_0 - \frac{\eta\nu}{64} \sqrt{\frac{i}{MB}} + 6\eta L \sum_{j=0}^{i-1} \mathbb{E}[|x_j - x_0|]. \quad (\text{B.132})$$

Next, we have the following lemma that we can apply to $\mathbb{E}[|x_j - x_0|]$. Proof of Lemma B.8 can be found in Appendix B.6.5.

Lemma B.8. For $x_0 \geq 0$, $0 \leq i \leq \frac{N}{B} - 1$ and $\eta \leq \frac{B}{513LN}$,

$$\mathbb{E}[|x_i - x_0|] \leq \frac{513}{512} \eta\nu \sqrt{\frac{i}{MB}} + \frac{513}{512} i\eta L x_0.$$

Applying this lemma to (B.132), we get

$$\begin{aligned}
\mathbb{E}[x_i | \mathcal{E} > 0] &\leq (1 - i\eta\mu)x_0 - \frac{\eta\nu}{64}\sqrt{\frac{i}{MB}} + \frac{1539\eta^2 L\nu}{256\sqrt{MB}} \sum_{j=0}^{i-1} \sqrt{j} + \frac{1539\eta^2 L^2 x_0}{256} \sum_{j=0}^{i-1} j \\
&\leq (1 - i\eta\mu)x_0 - \frac{\eta\nu}{64}\sqrt{\frac{i}{MB}} + \frac{513i^{3/2}\eta^2 L\nu}{128\sqrt{MB}} + \frac{1539i^2\eta^2 L^2}{512}x_0 \\
&= \left(1 - i\eta\mu + \frac{1539i^2\eta^2 L^2}{512}\right)x_0 - \left(\frac{1}{64} - \frac{513i\eta L}{128}\right)\eta\nu\sqrt{\frac{i}{MB}} \\
&\leq \left(1 - i\eta\mu + \frac{3i\eta L}{512}\right)x_0 - \frac{\eta\nu}{128}\sqrt{\frac{i}{MB}}. \tag{B.133}
\end{aligned}$$

where we got the last inequality by using the fact that $i\eta L \leq \frac{\eta LN}{B} \leq \frac{1}{513}$, which follows from $\eta \leq \frac{B}{513LN}$. So far, we have obtained an upper bound for $\mathbb{E}[x_i | \mathcal{E} > 0]$.

Recall that there is another conditional expectation in (B.127) that we want to bound, namely $\mathbb{E}[x_i | \mathcal{E} \leq 0]$. We bound it below, using the tools developed so far. For $i \leq \frac{N}{2B}$,

$$\begin{aligned}
\mathbb{E}[x_i | \mathcal{E} \leq 0] &= x_0 + \mathbb{E}[x_i - x_0 | \mathcal{E} \leq 0] \\
&\leq x_0 + \mathbb{E}[|x_i - x_0| | \mathcal{E} \leq 0] \\
&\leq x_0 + \frac{\mathbb{E}[|x_i - x_0|]}{\mathbb{P}(\mathcal{E} \leq 0)} \\
&\leq x_0 + 6\mathbb{E}[|x_i - x_0|] \tag{Using Lemma B.7} \\
&\leq x_0 + \frac{1539\eta\nu}{256}\sqrt{\frac{i}{MB}} + \frac{1539i\eta Lx_0}{256} \tag{Using Lemma B.8} \\
&\leq \left(1 + \frac{1539i\eta L}{256}\right)x_0 + \frac{1539\eta\nu}{256}\sqrt{\frac{i}{MB}} \tag{B.134}
\end{aligned}$$

Using (B.133) and (B.134) in (B.127), we get that for $i \leq \frac{N}{2B}$:

$$\begin{aligned}
&\mathbb{E}[(L1_{x_i \leq 0} + \mu 1_{x_i > 0})x_i] \\
&\leq \mathbb{P}(\mathcal{E} > 0) L \mathbb{E}[x_i | \mathcal{E} > 0] + \mathbb{P}(\mathcal{E} \leq 0) \mu \mathbb{E}[x_i | \mathcal{E} \leq 0] \\
&\leq \mathbb{P}(\mathcal{E} > 0) L \left(\left(1 - i\eta\mu + \frac{3i\eta L}{512}\right)x_0 - \frac{\eta\nu}{128}\sqrt{\frac{i}{MB}} \right) \\
&\quad + \mathbb{P}(\mathcal{E} \leq 0) \mu \left(\left(1 + \frac{1539i\eta L}{256}\right)x_0 + \frac{1539\eta\nu}{256}\sqrt{\frac{i}{MB}} \right). \tag{B.135}
\end{aligned}$$

From Lemma B.7, note that $\frac{1}{6} \leq \mathbb{P}(\mathcal{E} > 0) \leq \frac{5}{6}$ and $\frac{1}{6} \leq \mathbb{P}(\mathcal{E} \leq 0) \leq \frac{5}{6}$. We use these inequalities, along with $i\eta L \leq \frac{\eta LN}{B} \leq \frac{1}{513}$ and $\frac{L}{\mu} \geq 7695$, to bound the terms appearing in

(B.135).

$$\begin{aligned}
& \mathbb{P}(\mathcal{E} > 0) L \left(1 - i\eta\mu + \frac{3i\eta L}{512} \right) x_0 + \mathbb{P}(\mathcal{E} \leq 0) \mu \left(1 + \frac{1539i\eta L}{256} \right) x_0 \\
& \leq \frac{5}{6} L \left(1 + \frac{1}{87552} \right) x_0 + \frac{5}{6} \cdot \frac{L}{7695} \left(1 + \frac{3}{256} \right) x_0 \leq \frac{6}{7} L x_0.
\end{aligned} \tag{B.136}$$

We also have

$$\begin{aligned}
& -\mathbb{P}(\mathcal{E} > 0) \frac{\eta L \nu}{128} \sqrt{\frac{i}{MB}} + \mathbb{P}(\mathcal{E} \leq 0) \frac{1539\eta\mu\nu}{256} \sqrt{\frac{i}{MB}} \\
& \leq -\frac{\eta L \nu}{768} \sqrt{\frac{i}{MB}} + \frac{2565\eta\mu\nu}{512} \sqrt{\frac{i}{MB}} \\
& \leq -\frac{\eta L \nu}{1536} \sqrt{\frac{i}{MB}},
\end{aligned} \tag{B.137}$$

where we used the assumption $\frac{L}{\mu} \geq 7695$. Substituting (B.136) and (B.137) to (B.135), we get

$$\mathbb{E}[(L1_{x_i \leq 0} + \mu 1_{x_i > 0})x_i] \leq \frac{6}{7} L x_0 - \frac{\eta L \nu}{1536} \sqrt{\frac{i}{MB}},$$

as desired.

B.6.2 Proof of Lemma B.5

$$\begin{aligned}
\mathbb{E}[(L1_{x_i \leq 0} + \mu 1_{x_i > 0})x_i] & \leq \mu \mathbb{E}[x_i] \\
& = \mu x_0 + \mu \mathbb{E}[x_i - x_0] \\
& \leq \mu x_0 + \mu \mathbb{E}[|x_i - x_0|] \\
& \leq \mu x_0 + \frac{513}{512} i \eta L \mu x_0 + \frac{513}{512} \eta \mu \nu \sqrt{\frac{i}{MB}}. \quad (\text{Using Lemma B.8})
\end{aligned}$$

B.6.3 Proof of Lemma B.6

We consider iterates within a single epoch, and hence we will omit the subscripts denoting epochs. In our construction, each machine has the same set of component functions, that is, there will be no inter-machine deviation. We therefore omit the superscript m from the local component functions. Consider the function

$$G_2(x) := \frac{1}{N} \left(\sum_{i=1}^{\frac{N}{2}} g_{+1}(x) + \sum_{i=\frac{N}{2}+1}^N g_{-1}(x) \right), \text{ where}$$

$$g_{+1}(x) := \frac{Lx^2}{2} + \nu x, \text{ and } g_{-1}(x) := \frac{Lx^2}{2} - \nu x.$$

Hence, $G_2(x) = \frac{Lx^2}{2}$. We will prove the lemma by coupling iterates corresponding to F_2 and G_2 . In particular, we will perform minibatch RR on F_2 and G_2 such that both start the given epoch at x_0 and all the corresponding machines use the same random permutations. Let $x_{i,F}$ be the iterate after the i -th round of communication for F_2 and $x_{i,G}$ be the iterate after the i -th round of communication for G_2 . We use mathematical induction to prove that $x_{i,F} \geq x_{i,G}$ for all $i = 0, \dots, \frac{N}{B}$. After that, we will use this to prove our desired statement $\mathbb{E}[x_{\frac{N}{B},F} \mid x_0 < 0] \geq (1 - \frac{7\eta LN}{8B})x_0$. Let σ^m be a random permutation of $\frac{N}{2} + 1$'s and $\frac{N}{2} - 1$'s.

Base case. $x_{0,F} \geq x_{0,G}$ since both start the epoch at the same point x_0 .

Inductive case. There can be three cases:

- Case 1: $x_{i,F} \geq x_{i,G} \geq 0$. Then,

$$\begin{aligned} x_{i+1,F} - x_{i+1,G} &= x_{i,F} - x_{i,G} - \frac{\eta}{MB} \sum_{m=1}^M \sum_{j=iB+1}^{(i+1)B} (\nabla f_{\sigma_j^m}(x_{i,F}) - \nabla g_{\sigma_j^m}(x_{i,G})) \\ &= x_{i,F} - x_{i,G} - \frac{\eta}{MB} \sum_{m=1}^M \sum_{j=iB+1}^{(i+1)B} (\mu x_{i,F} + \nu \sigma_j^m - Lx_{i,G} - \nu \sigma_j^m) \\ &= x_{i,F} - x_{i,G} - \eta (\mu x_{i,F} - Lx_{i,G}) \\ &= x_{i,F}(1 - \eta\mu) - x_{i,G}(1 - \eta L) \\ &\geq 0. \end{aligned}$$

- Case 2: $0 \geq x_{i,F} \geq x_{i,G}$. Then,

$$\begin{aligned} x_{i+1,F} - x_{i+1,G} &= x_{i,F} - x_{i,G} - \frac{\eta}{MB} \sum_{m=1}^M \sum_{j=iB+1}^{(i+1)B} (\nabla f_{\sigma_j^m}(x_{i,F}) - \nabla g_{\sigma_j^m}(x_{i,G})) \\ &= x_{i,F} - x_{i,G} - \frac{\eta}{MB} \sum_{m=1}^M \sum_{j=iB+1}^{(i+1)B} (Lx_{i,F} + \nu \sigma_j^m - Lx_{i,G} - \nu \sigma_j^m) \\ &= x_{i,F} - x_{i,G} - \eta (Lx_{i,F} - Lx_{i,G}) \\ &= x_{i,F}(1 - \eta L) - x_{i,G}(1 - \eta L) \\ &\geq 0. \end{aligned}$$

- Case 3: $x_{i,F} \geq 0 \geq x_{i,G}$. Then,

$$\begin{aligned}
x_{i+1,F} - x_{i+1,G} &= x_{i,F} - x_{i,G} - \frac{\eta}{MB} \sum_{m=1}^M \sum_{j=iB+1}^{(i+1)B} (\nabla f_{\sigma_j^m}(x_{i,F}) - \nabla g_{\sigma_j^m}(x_{i,G})) \\
&= x_{i,F} - x_{i,G} - \frac{\eta}{MB} \sum_{m=1}^M \sum_{j=iB+1}^{(i+1)B} (\mu x_{i,F} + \nu \sigma_j^m - L x_{i,G} - \nu \sigma_j^m) \\
&= x_{i,F} - x_{i,G} - \eta (\mu x_{i,F} - L x_{i,G}) \\
&= x_{i,F}(1 - \eta\mu) - x_{i,G}(1 - \eta L).
\end{aligned}$$

Note that since $\eta \leq \frac{B}{LN}$, $x_{i,F}(1 - \eta\mu) \geq 0$ and $x_{i,G}(1 - \eta L) \leq 0$, which proves that $x_{i+1,F} - x_{i+1,G} \geq 0$.

Thus, we see that $x_{i+1,F} \geq x_{i+1,G}$. Further, by linearity of expectation and gradient, it is easy to check that

$$\begin{aligned}
\mathbb{E}[x_{\frac{N}{B},G}] &= \mathbb{E}[x_{\frac{N}{B}-1,G} - \eta \nabla G_2(x_{\frac{N}{B}-1,G})] \\
&= (1 - \eta L) \mathbb{E}[x_{\frac{N}{B}-1,G}] \\
&= \dots \\
&= (1 - \eta L)^{\frac{N}{B}} x_0.
\end{aligned}$$

Using the result that $x_{\frac{N}{B},F} \geq x_{\frac{N}{B},G}$ which we proved above, we get $\mathbb{E}[x_{\frac{N}{B},F}] \geq (1 - \eta L)^{\frac{N}{B}} x_0$ for any initial iterate x_0 . Specifically for $x_0 < 0$, this implies $\mathbb{E}[x_{\frac{N}{B},F} \mid x_0 < 0] \geq (1 - \eta L)^{\frac{N}{B}} x_0$.

Further, since $\eta L \leq \frac{B}{513N}$, we have $(1 - \eta L)^{\frac{N}{B}} \geq 1 - \frac{7\eta LN}{8B}$. This is because $1 - \frac{7zN}{8B} - (1 - z)^{\frac{N}{B}}$ is nonnegative on the interval $\left[0, 1 - (7/8)^{\frac{1}{N/B-1}}\right]$, and $1 - (7/8)^{\frac{1}{N/B-1}} \geq \frac{B}{513N}$ for all $\frac{N}{B} \geq 2$. To see why, note that $(1 - \frac{1}{513(n-1)})^{n-1} \geq \frac{7}{8}$ for all $n \geq 2$, and this gives $1 - (7/8)^{\frac{1}{n-1}} \geq \frac{1}{513(n-1)}$, which then implies $1 - (7/8)^{\frac{1}{n-1}} \geq \frac{1}{513n}$ for all $n \geq 2$. Therefore, for $x_0 < 0$, we have $\mathbb{E}[x_{\frac{N}{B},F} \mid x_0 < 0] \geq (1 - \frac{7\eta LN}{8B}) x_0$.

For the last statement of the lemma, note that by symmetry of the function G_2 , if we initialize the Algorithm 5.2 at 0, then for any starting iterate of an epoch we have $\mathbb{P}(x_{0,G} \geq 0) \geq 1/2$. This combined with the fact that $x_{i,F} \geq x_{i,G}$ gives us that $\mathbb{P}(x_{0,F} \geq 0) \geq 1/2$.

B.6.4 Proof of Lemma B.7

For $m = 1, \dots, M$, let σ^m be a random permutation of $\frac{N}{2} + 1$'s and $\frac{N}{2} - 1$'s. Then, we first show that for any $i \leq N/2$ and $k \leq B/2$,

$$\frac{1}{64} \left(\sqrt{\frac{i}{M}} + \sqrt{k} \right) \leq \mathbb{E} \left[\left| \left(\frac{1}{M} \sum_{m=1}^M \sum_{j=1}^i \sigma_j^m \right) + \sum_{j=i+1}^{i+k} \sigma_j^M \right| \right].$$

To prove the lower bound, we will use Khintchine's inequality along with Lemma 12 from Rajput et al. (2020). Let us define random variables $\mathbf{a}_m := |\frac{1}{M} \sum_{j=1}^i \sigma_j^m|$, $\mathbf{x}_m := \text{sign}(\sum_{j=1}^i \sigma_j^m)$, $\mathbf{b}_M := |\sum_{j=i+1}^{i+k} \sigma_j^M|$, and $\mathbf{y}_M := \text{sign}(\sum_{j=i+1}^{i+k} \sigma_j^M)$. For $\mathbf{x}_m$, if the sum $\sum_{j=1}^i \sigma_j^m = 0$ then $\mathbf{x}_m$ is $+1$ with probability 0.5 and -1 with probability 0.5. Ties occurring in $\mathbf{y}_M$ are also broken similarly. We can note that $\mathbf{x}_m$'s and $\mathbf{y}_M$ are i.i.d. Rademacher random variables, which allows us to apply Khintchine's inequality. Then, by Khintchine's inequality,

$$\begin{aligned} \mathbb{E} \left[\left| \left(\frac{1}{M} \sum_{m=1}^M \sum_{j=1}^i \sigma_j^m \right) + \sum_{j=i+1}^{i+k} \sigma_j^M \right| \right] &= \mathbb{E} \left[\left| \sum_{m=1}^M \mathbf{a}_m \mathbf{x}_m + \mathbf{b}_M \mathbf{y}_M \right| \right] \\ &\geq \frac{1}{\sqrt{2}} \mathbb{E} \left[\left(\sum_{m=1}^M \mathbf{a}_m^2 + \mathbf{b}_M^2 \right)^{1/2} \right]. \end{aligned}$$

By applying $\|z\|_2 \geq \frac{1}{\sqrt{d}} \|z\|_1$ for $z \in \mathbb{R}^d$ twice, we get

$$\frac{1}{\sqrt{2}} \mathbb{E} \left[\left(\sum_{m=1}^M \mathbf{a}_m^2 + \mathbf{b}_M^2 \right)^{1/2} \right] \geq \frac{1}{2} \mathbb{E} \left[\left(\sum_{m=1}^M \mathbf{a}_m^2 \right)^{1/2} + \mathbf{b}_M \right] \geq \frac{1}{2} \mathbb{E} \left[\frac{1}{\sqrt{M}} \sum_{m=1}^M \mathbf{a}_m + \mathbf{b}_M \right].$$

Next, noticing that $\mathbf{a}_m$'s are i.i.d.,

$$\begin{aligned} \frac{1}{2} \mathbb{E} \left[\frac{1}{\sqrt{M}} \sum_{m=1}^M \mathbf{a}_m + \mathbf{b}_M \right] &= \frac{1}{2} \mathbb{E} \left[\sqrt{M} \mathbf{a}_M + \mathbf{b}_M \right] = \frac{1}{2} \mathbb{E} \left[\frac{1}{\sqrt{M}} \left| \sum_{j=1}^i \sigma_j^M \right| + \left| \sum_{j=i+1}^{i+k} \sigma_j^M \right| \right] \\ &\geq \frac{1}{64} \left(\sqrt{\frac{i}{M}} + \sqrt{k} \right). \quad (\text{Lemma 12 from Rajput et al. (2020)}) \end{aligned}$$

Note that Lemma 12 from Rajput et al. (2020) has the requirement that $N \geq 256$. However, that requirement is for the entire lemma to hold, whereas we need only the first inequality in the lemma. For that, the requirement is simply $N \geq 8$. Further, note that for $N = 2, 4$, and 6 it can be manually verified that the required inequalities in Lemma 12 of Rajput et al. (2020) hold. Hence, this lemma holds for all even N .

The upper bound comes from Jensen's inequality:

$$\begin{aligned}
\mathbb{E} \left[\left\| \left(\frac{1}{M} \sum_{m=1}^M \sum_{j=1}^i \sigma_j^m \right) + \sum_{j=i+1}^{i+k} \sigma_j^M \right\|^2 \right] &\leq \frac{1}{M} \mathbb{E} \left[\left\| \sum_{m=1}^M \sum_{j=1}^i \sigma_j^m \right\|^2 \right] + \mathbb{E} \left[\left\| \sum_{j=i+1}^{i+k} \sigma_j^M \right\|^2 \right] \\
&\leq \frac{1}{M} \sqrt{\mathbb{E} \left[\left(\sum_{m=1}^M \sum_{j=1}^i \sigma_j^m \right)^2 \right]} + \sqrt{\mathbb{E} \left[\left(\sum_{j=i+1}^{i+k} \sigma_j^M \right)^2 \right]} \\
&= \frac{1}{M} \sqrt{\sum_{m=1}^M \mathbb{E} \left[\left(\sum_{j=1}^i \sigma_j^m \right)^2 \right]} + \sqrt{\mathbb{E} \left[\left(\sum_{j=i+1}^{i+k} \sigma_j^M \right)^2 \right]} \\
&\quad \text{(Since } \sum_{j=1}^i \sigma_j^m \text{ for } m = 1, 2, \dots \text{ are mean 0 and independent.)} \\
&= \sqrt{\frac{1}{M} \mathbb{E} \left[\left(\sum_{j=1}^i \sigma_j^M \right)^2 \right]} + \sqrt{\mathbb{E} \left[\left(\sum_{j=i+1}^{i+k} \sigma_j^M \right)^2 \right]} \\
&= \sqrt{\frac{1}{M} \left(i + \sum_{j \neq l} \mathbb{E}[\sigma_j^M \sigma_l^M] \right)} + \sqrt{k + \sum_{j \neq l} \mathbb{E}[\sigma_j^M \sigma_l^M]}.
\end{aligned}$$

$\mathbb{E}[\sigma_j^M \sigma_l^M] \leq 0$ because σ_j^M and σ_l^M are negatively correlated. Hence, we get

$$\mathbb{E} \left[\left\| \left(\frac{1}{M} \sum_{m=1}^M \sum_{j=1}^i \sigma_j^m \right) + \sum_{j=i+1}^{i+k} \sigma_j^M \right\|^2 \right] \leq \sqrt{\frac{i}{M}} + \sqrt{k},$$

as desired.

Next, it is left to show that for $0 \leq i \leq \frac{N}{2}$ and $0 \leq k \leq \frac{B}{2}$ satisfying $i + k \geq 1$, we have

$$\mathbb{P} \left(\sum_{m=1}^M \sum_{j=1}^i \sigma_j^m + M \sum_{j=i+1}^{i+k} \sigma_j^M > 0 \right) = \mathbb{P} \left(\sum_{m=1}^M \sum_{j=1}^i \sigma_j^m + M \sum_{j=i+1}^{i+k} \sigma_j^M < 0 \right) \geq \frac{1}{6}.$$

By symmetry, proving the equality is straightforward, and hence it is sufficient prove that

$$\mathbb{P} \left(\sum_{m=1}^M \sum_{j=1}^i \sigma_j^m + M \sum_{j=i+1}^{i+k} \sigma_j^M = 0 \right) \leq \frac{2}{3}. \tag{B.138}$$

For this, it in fact suffices to show that

$$\mathbb{P} \left(\sum_{j=1}^l \sigma_j^1 = 0 \right) \leq \frac{2}{3} \text{ for all } 1 \leq l \leq N-1, \tag{B.139}$$

because (B.138) can be derived from (B.139). We first explain why (B.139) implies (B.138), and then show (B.139).

Suppose (B.139) is true. Then,

- Case 1: If $i = 0$, then (B.138) becomes $\mathbb{P}(\sum_{j=1}^k \sigma_j^M = 0) \leq \frac{2}{3}$, which is true due to (B.139).
- Case 2: If $M = 1$, then (B.138) becomes $\mathbb{P}(\sum_{j=1}^{i+k} \sigma_j^1 = 0) \leq \frac{2}{3}$, which is true due to (B.139).
- Case 3: If $i \geq 1$ and $M \geq 2$, then we can consider two events that partition the probability space:

1. $E_1 := \{\sum_{m=2}^M \sum_{j=1}^i \sigma_j^m + M \sum_{j=i+1}^{i+k} \sigma_j^M = 0\}$. Conditioned on this event E_1 ,

$$\mathbb{P}\left(\sum_{m=1}^M \sum_{j=1}^i \sigma_j^m + M \sum_{j=i+1}^{i+k} \sigma_j^M = 0 \mid E_1\right) = \mathbb{P}\left(\sum_{j=1}^i \sigma_j^1 = 0\right) \leq \frac{2}{3}$$

due to independence of machines and (B.139).

2. $E_2 := \{\sum_{m=2}^M \sum_{j=1}^i \sigma_j^m + M \sum_{j=i+1}^{i+k} \sigma_j^M \neq 0\}$. Conditioned on this event E_2 , let $c := \sum_{m=2}^M \sum_{j=1}^i \sigma_j^m + M \sum_{j=i+1}^{i+k} \sigma_j^M$. Then,

$$\mathbb{P}\left(\sum_{m=1}^M \sum_{j=1}^i \sigma_j^m + M \sum_{j=i+1}^{i+k} \sigma_j^M = 0 \mid E_2\right) = \mathbb{P}\left(\sum_{j=1}^i \sigma_j^1 = -c\right).$$

However, by symmetry, $\mathbb{P}\left(\sum_{j=1}^i \sigma_j^1 = -c\right) = \mathbb{P}\left(\sum_{j=1}^i \sigma_j^1 = c\right) \leq \frac{1}{2}$.

From these two events, we conclude that (B.138) must hold.

It is now left to prove (B.139). It is clear that $\mathbb{P}(\sum_{j=1}^i \sigma_j^1 = 0) = 0$ for all odd i , so we assume that i is even. Also note that $\mathbb{P}(\sum_{j=1}^i \sigma_j^1 = 0) = \mathbb{P}(\sum_{j=i+1}^N \sigma_j^1 = 0) = \mathbb{P}(\sum_{j=1}^{N-i} \sigma_j^1 = 0)$, since σ_j^1 's sum to zero. Therefore, for the rest of the proof, we can focus on even i 's in the range $2 \leq i \leq \frac{N}{2}$. Note that $\mathbb{P}(\sum_{j=1}^i \sigma_j^m = 0)$ is just the probability of having $\frac{i}{2} + 1$'s and $\frac{i}{2} - 1$'s in the first i spots in a random shuffling of $\frac{N}{2} + 1$'s and $\frac{N}{2} - 1$'s. This is equivalent to choosing $\frac{i}{2}$ indices (for $+1$) out of the first i , and then choosing $\frac{N-i}{2}$ indices out of the remaining $N - i$. Thus,

$$\mathbb{P}\left(\sum_{j=1}^i \sigma_j^m = 0\right) = \frac{\binom{i}{i/2} \cdot \binom{N-i}{\frac{N-i}{2}}}{\binom{N}{N/2}}.$$

Note that the term above is a decreasing function of i for $i \leq N/2$. Hence, putting $i = 2$ to the RHS we get

$$\mathbb{P}\left(\sum_{j=1}^i \sigma_j^m = 0\right) \leq \frac{\binom{2}{1} \cdot \binom{N-2}{\frac{N-2}{2}}}{\binom{N}{N/2}} = \frac{N}{2(N-1)} \leq \frac{2}{3},$$

where the inequality holds for $N \geq 4$.

B.6.5 Proof of Lemma B.8

$$\begin{aligned} \mathbb{E}[|x_i - x_0|] &= \mathbb{E}\left[\left|\frac{\eta}{MB} \sum_{j=0}^{i-1} \sum_{m=1}^M \sum_{k=jB+1}^{(j+1)B} \nu \sigma_k^m + (L1_{x_j < 0} + \mu 1_{x_j \geq 0})x_j\right|\right] \\ &\leq \frac{\eta\nu}{B} \sqrt{\frac{iB}{M}} + \eta \mathbb{E}\left[\left|\sum_{j=0}^{i-1} (L1_{x_j < 0} + \mu 1_{x_j \geq 0})x_j\right|\right] \quad (\text{By Lemma B.7}) \\ &\leq \eta\nu \sqrt{\frac{i}{MB}} + \eta L \sum_{j=0}^{i-1} \mathbb{E}[|x_j|] \\ &\leq \eta\nu \sqrt{\frac{i}{MB}} + i\eta L x_0 + \eta L \sum_{j=0}^{i-1} \mathbb{E}[|x_j - x_0|]. \end{aligned}$$

Let $h(i) := \eta\nu \sqrt{\frac{i}{MB}} + i\eta L x_0 + \eta L \sum_{j=0}^{i-1} h(j)$, starting with $h(0) = 0$. Then using induction, it can be seen that $\mathbb{E}[|x_i - x_0|] \leq h(i)$. Further, since $h(i)$ is an increasing function of i , we get

$$h(i) \leq \frac{\eta\nu \sqrt{\frac{i}{MB}} + i\eta L x_0}{1 - i\eta L}.$$

Since $i \leq \frac{N}{B}$ and $\eta \leq \frac{B}{513LN}$, we get that $\frac{1}{1-i\eta L} \leq \frac{513}{512}$, so $\mathbb{E}[|x_i - x_0|] \leq \frac{513}{512} \eta\nu \sqrt{\frac{i}{MB}} + \frac{513}{512} i\eta L x_0$.

B.7 Proof of lower bound for local RR: homogeneous case (Theorem 5.10)

Recall that Theorem 5.10 gives the bound for local RR in the homogeneous setting, where all machines have the same local objectives. Similar to Theorem 5.8, we consider three step-size ranges and do case analysis for each of them. We construct functions for each corresponding step-size regime such that the convergence of local RR is “slow” for the functions on their corresponding step-size regime. The final lower bound is the minimum among the lower bounds obtained for the three regimes. More concretely, we will construct

three one-dimensional functions $F_1(x)$, $F_2(x)$, and $F_3(x)$ satisfying L -smoothness (5.1), μ -PL condition (5.2), and Assumption 5.2 such that⁸

- Local RR on $F_1(x)$ with $\eta \leq \frac{1}{\mu NK}$ and initialization $y_0 = \frac{\nu}{\mu}$ results in

$$\mathbb{E}[F_1(y_{K, \frac{N}{B}})] = \Omega\left(\frac{\nu^2}{\mu}\right).$$

- Local RR on $F_2(x)$ with $\eta \geq \frac{1}{\mu NK}$ and $\eta \leq \frac{1}{1025LN}$ and initialization $y_0 = 0$ results in

$$\mathbb{E}[F_2(y_{K, \frac{N}{B}})] = \Omega\left(\frac{\nu^2}{\mu MNK^2} + \frac{\nu^2 B}{\mu N^2 K^2}\right).$$

Note that the step-size range requires $K \geq 1025\kappa$, hence this lower bound occurs only in the “large-epoch” regime, i.e., $K \gtrsim \kappa$.

- Local RR on $F_3(x)$ with $\eta \geq \frac{1}{\mu NK}$ and $\eta \geq \frac{1}{1025LN}$ and initialization $y_0 = 0$ results in

$$\mathbb{E}[F_3(y_{K, \frac{N}{B}})] = \Omega\left(\frac{\nu^2}{\mu MNK}\right).$$

Then, the three dimensional function $F([x, y, z]^\top) = F_1(x) + F_2(y) + F_3(z)$ will show bad convergence in any step-size regime. Furthermore,

$$\mu \mathbf{I} \preceq \min(\nabla^2 F_1, \nabla^2 F_2, \nabla^2 F_3) \mathbf{I} \preceq \nabla^2 F \preceq \max(\nabla^2 F_1, \nabla^2 F_2, \nabla^2 F_3) \mathbf{I} \preceq L \mathbf{I},$$

that is, if F_1 , F_2 and F_3 are μ -strongly convex and L -smooth, then so is F . Moreover, since the component functions in each coordinate are designed to satisfy Assumption 5.2 with ν , the resulting three dimensional function F also satisfies Assumption 5.2 with $\sqrt{3}\nu$.

Since the final lower bound is the minimum among the lower bounds obtained in the step-size ranges, the lower bound becomes $\Omega\left(\frac{\nu^2}{\mu MNK^2} + \frac{\nu^2 B}{\mu N^2 K^2}\right)$ if $K \geq 1025\kappa$ and $K \geq \frac{MB}{N}$ (this inequality is required to make sure $\frac{\nu^2 B}{\mu N^2 K^2} \leq \frac{\nu^2}{\mu MNK}$) and $\Omega\left(\frac{\nu^2}{\mu MNK}\right)$ otherwise.

In the subsequent subsections, we prove the lower bounds for F_1 , F_2 , and F_3 separately.

B.7.1 Lower bound for $\eta \leq \frac{1}{\mu NK}$

Consider the case where every function at every machine is the same: for all $i \in [N]$ and $m \in [M]$, $f_i^m(x) := \frac{\mu x^2}{2}$. Hence, $F_1(x) = \frac{\mu x^2}{2}$.

⁸Again, the functions constructed in this theorem are μ -strongly convex, which is stronger than μ -PL required in Definition 5.5. Also, our functions satisfy Assumption 5.3 with $\tau = 0$, $\rho = 1$.

Since all $f_i^{m'}$'s are the same, the local updates in all the machines are identical. Hence, for this subsection we omit the superscript for local machines. Let $x_{k,0}$ and $x_{k,N}$ denote the local iterates at the beginning and end of the k -th epoch. Then,

$$x_{k,N} = (1 - \eta\mu)^N x_{k,0}.$$

Initializing at $x_{1,0} = \frac{\nu}{\mu}$ and repeating this for K epochs, we get that after K epochs, the last iterate $y_{K, \frac{N}{B}} = x_{K,N}$ satisfies

$$y_{K, \frac{N}{B}} = x_{K,N} = (1 - \eta\mu)^{NK} \cdot \frac{\nu}{\mu} \geq \left(1 - \frac{1}{NK}\right)^{NK} \cdot \frac{\nu}{\mu} \geq \frac{\nu}{4\mu},$$

since $N \geq 2$, and $K \geq 1$. Hence, $F_1(y_{K, \frac{N}{B}}) = \Omega(\frac{\nu^2}{\mu})$.

B.7.2 Lower bound for $\eta \geq \frac{1}{\mu NK}$ and $\eta \leq \frac{1}{1025LN}$

For most part of this subsection, we consider iterates within a single epoch, and hence we will omit the subscripts denoting epochs. Let x_0 denote the iterate at the beginning of the epoch (which is the same across all the machines), and x_i^m denote the i -th local iterate for machine m . After every B local iterates, the server aggregates the local iterates x_{iB}^m , computes their average $y_i := \frac{1}{M} \sum_{m=1}^M x_{iB}^m$, and synchronizes all the machines $x_{iB}^m := y_i$.

Let y_0 denote the iterate at the beginning of the epoch (which is the same across all the machines $x_0^m = y_0$), and x_i^m denote the i -th local iterate at machine m . In our construction, each machine will have the same set of component functions, that is, there will be no inter-machine deviation. We therefore omit the superscript m from the local component functions f_i^m . The function we construct for the lower bound and its component functions are as follows:

$$F_2(x) := \frac{1}{N} \left(\sum_{i=1}^{\frac{N}{2}} f_{+1}(x) + \sum_{i=\frac{N}{2}+1}^N f_{-1}(x) \right), \text{ where}$$

$$f_{+1}(x) := (L1_{x \leq 0} + \mu 1_{x > 0}) \frac{x^2}{2} + \nu x, \text{ and}$$

$$f_{-1}(x) := (L1_{x \leq 0} + \mu 1_{x > 0}) \frac{x^2}{2} - \nu x$$

Note that the function $F_2(x) = (L1_{x \leq 0} + \mu 1_{x > 0}) \frac{x^2}{2}$ is μ -strongly convex and L -smooth with minimizer at 0, and also satisfies Assumption 5.2.

Let σ^m be a random permutation of $\frac{N}{2} + 1$'s and $\frac{N}{2} - 1$'s. Then, machine m computes

gradients on f_{-1} and f_{+1} in the order given by σ^m . Let σ_j^m denote the j -th ordered element of σ^m . Then,

$$\nabla f_{\sigma_j^m}(x) = (L1_{x \leq 0} + \mu 1_{x > 0})x + \nu \sigma_j^m.$$

Hence, the last iterate of an epoch, $y_{\frac{N}{B}}$, is given by

$$\begin{aligned} y_{\frac{N}{B}} - y_0 &= \sum_{i=0}^{\frac{N}{B}-1} \left(-\frac{\eta}{M} \sum_{m=1}^M \sum_{j=0}^{B-1} \nabla f_{\sigma_{iB+j+1}^m}(x_{iB+j}^m) \right) \\ &= -\frac{\eta}{M} \sum_{i=0}^{\frac{N}{B}-1} \sum_{m=1}^M \sum_{j=0}^{B-1} ((L1_{x_{iB+j}^m \leq 0} + \mu 1_{x_{iB+j}^m > 0})x_{iB+j}^m + \nu \sigma_{iB+j+1}^m) \\ &= -\frac{\eta}{M} \sum_{i=0}^{\frac{N}{B}-1} \sum_{m=1}^M \sum_{j=0}^{B-1} (L1_{x_{iB+j}^m \leq 0} + \mu 1_{x_{iB+j}^m > 0})x_{iB+j}^m, \end{aligned}$$

where, in the last line, we used the fact that $\sum_{j=1}^N \sigma_j^m = 0$.

Recall that in the construction, each machine has the same component functions. Hence,

$$\begin{aligned} \mathbb{E}[y_{\frac{N}{B}} - y_0] &= -\frac{\eta}{M} \mathbb{E} \left[\sum_{i=0}^{\frac{N}{B}-1} \sum_{m=1}^M \sum_{j=0}^{B-1} (L1_{x_{iB+j}^m \leq 0} + \mu 1_{x_{iB+j}^m > 0})x_{iB+j}^m \right] \\ &= -\eta \sum_{i=0}^{\frac{N}{B}-1} \sum_{j=0}^{B-1} \mathbb{E} [(L1_{x_{iB+j}^1 \leq 0} + \mu 1_{x_{iB+j}^1 > 0})x_{iB+j}^1], \end{aligned} \quad (\text{B.140})$$

where the last equality holds because the iterates x_{iB+j}^m are identically distributed across different $m \in [M]$. Hence, we need to bound $\mathbb{E}[(L1_{x_{iB+j}^1 \leq 0} + \mu 1_{x_{iB+j}^1 > 0})x_{iB+j}^1]$. As we did for Theorem 5.8, we want to prove that $\mathbb{E}[y_{\frac{N}{B}}]$ keeps increasing over an epoch, that is $\mathbb{E}[y_{\frac{N}{B}} - y_0] > 0$ when y_0 is close enough to the minimizer 0.

For this, we first consider the case where the first iterate y_0 of the epoch satisfies $y_0 \geq 0$. The $y_0 < 0$ case will be considered later. For the case $y_0 \geq 0$, we will show that whenever y_0 is small, the expected amount of update made in the $(iB + j + 1)$ -th iteration, $\mathbb{E}[(L1_{x_{iB+j}^1 \leq 0} + \mu 1_{x_{iB+j}^1 > 0})x_{iB+j}^1]$, is negative if $i \leq \lfloor \frac{N}{2B} \rfloor$ and $\frac{B}{4} \leq j \leq \frac{B}{2}$, and not too big otherwise.

We use the following lemmas, proven in Appendices B.8.1 and B.8.2, respectively.

Lemma B.9. For $y_0 \geq 0$, $0 \leq i \leq \lfloor \frac{N}{2B} \rfloor$, $\frac{B}{4} \leq j \leq \frac{B}{2}$, $\eta \leq \frac{1}{1025LN}$, and $\frac{L}{\mu} \geq \frac{15375}{2}$,

$$\mathbb{E}[(L1_{x_{iB+j}^1 \leq 0} + \mu 1_{x_{iB+j}^1 > 0})x_{iB+j}^1] \leq \frac{6}{7}Ly_0 - \frac{\eta L\nu}{1536} \left(\sqrt{\frac{iB}{M}} + \sqrt{j} \right).$$

Lemma B.10. For $y_0 \geq 0$, $0 \leq i \leq \frac{N}{B} - 1$, $0 \leq j \leq B - 1$, and $\eta \leq \frac{1}{1025LN}$,

$$\begin{aligned} \mathbb{E}[(L1_{x_{iB+j}^1 \leq 0} + \mu 1_{x_{iB+j}^1 > 0})x_{iB+j}^1] &\leq \mu \left(1 + \frac{1025(iB+j)\eta L}{1024} \right) y_0 \\ &\quad + \frac{1025\eta\mu\nu}{1024} \left(\sqrt{\frac{iB}{M}} + \sqrt{j} \right). \end{aligned}$$

Next, we apply the two lemmas above in (B.140). For now, we consider the case $N/B \geq 2$. The case $B = N$ will be handled separately at the end of this subsection. For simplicity of notation, define $E_{iB+j} := \mathbb{E}[(L1_{x_{iB+j}^1 \leq 0} + \mu 1_{x_{iB+j}^1 > 0})x_{iB+j}^1]$. We will divide the summation in (B.140) into four groups; for one of them we can apply Lemma B.9, and for the other three we apply Lemma B.10.

$$\begin{aligned} \mathbb{E}[y_{\frac{N}{B}} - y_0] &= -\eta \sum_{i=0}^{\frac{N}{B}-1} \sum_{j=0}^{B-1} E_{iB+j} \\ &= -\eta \left(\sum_{i=0}^{\lfloor \frac{N}{2B} \rfloor} \sum_{j=0}^{\frac{B}{4}-1} E_{iB+j} + \sum_{i=0}^{\lfloor \frac{N}{2B} \rfloor} \sum_{j=\frac{B}{4}}^{\frac{B}{2}} E_{iB+j} + \sum_{i=0}^{\lfloor \frac{N}{2B} \rfloor} \sum_{j=\frac{B}{2}+1}^{B-1} E_{iB+j} + \sum_{i=\lfloor \frac{N}{2B} \rfloor+1}^{\frac{N}{B}-1} \sum_{j=0}^{B-1} E_{iB+j} \right) \\ &\geq -\eta \sum_{i=0}^{\lfloor \frac{N}{2B} \rfloor} \sum_{j=0}^{\frac{B}{4}-1} \left(\mu \left(1 + \frac{1025(iB+j)\eta L}{1024} \right) y_0 + \frac{1025\eta\mu\nu}{1024} \left(\sqrt{\frac{iB}{M}} + \sqrt{j} \right) \right) \\ &\quad - \eta \sum_{i=0}^{\lfloor \frac{N}{2B} \rfloor} \sum_{j=\frac{B}{4}}^{\frac{B}{2}} \left(\frac{6}{7}Ly_0 - \frac{\eta L\nu}{1536} \left(\sqrt{\frac{iB}{M}} + \sqrt{j} \right) \right) \\ &\quad - \eta \sum_{i=0}^{\lfloor \frac{N}{2B} \rfloor} \sum_{j=\frac{B}{2}+1}^{B-1} \left(\mu \left(1 + \frac{1025(iB+j)\eta L}{1024} \right) y_0 + \frac{1025\eta\mu\nu}{1024} \left(\sqrt{\frac{iB}{M}} + \sqrt{j} \right) \right) \\ &\quad - \eta \sum_{i=\lfloor \frac{N}{2B} \rfloor+1}^{\frac{N}{B}-1} \sum_{j=0}^{B-1} \left(\mu \left(1 + \frac{1025(iB+j)\eta L}{1024} \right) y_0 + \frac{1025\eta\mu\nu}{1024} \left(\sqrt{\frac{iB}{M}} + \sqrt{j} \right) \right) \\ &\geq -\eta \sum_{i=0}^{\lfloor \frac{N}{2B} \rfloor} \sum_{j=\frac{B}{4}}^{\frac{B}{2}} \left(\frac{6}{7}Ly_0 - \frac{\eta L\nu}{1536} \left(\sqrt{\frac{iB}{M}} + \sqrt{j} \right) \right) \\ &\quad - \eta \sum_{i=0}^{\frac{N}{B}-1} \sum_{j=0}^{B-1} \left(\mu \left(1 + \frac{1025(iB+j)\eta L}{1024} \right) y_0 + \frac{1025\eta\mu\nu}{1024} \left(\sqrt{\frac{iB}{M}} + \sqrt{j} \right) \right), \tag{B.141} \end{aligned}$$

where the last inequality is true because the RHS of the inequality in Lemma B.10 is nonnegative. First consider the terms in (B.141) that involve y_0 . Since $(iB + j)\eta L \leq \eta LN \leq \frac{1}{1025}$, $\mu \leq \frac{2L}{15375}$, $N/B \geq 2$, and $B \geq 4$, we have the following loose bound:

$$\begin{aligned}
& \sum_{i=0}^{\lfloor \frac{N}{2B} \rfloor} \sum_{j=\frac{B}{4}}^{\frac{B}{2}} \frac{6}{7}L + \sum_{i=0}^{\frac{N}{B}-1} \sum_{j=0}^{B-1} \mu \left(1 + \frac{1025(iB + j)\eta L}{1024} \right) \\
& \leq \left(\left\lfloor \frac{N}{2B} \right\rfloor + 1 \right) \left(\frac{B}{4} + 1 \right) \frac{6}{7}L + \frac{N}{B} \cdot B \cdot \frac{2L}{15375} \left(1 + \frac{1}{1024} \right) \\
& \leq \frac{N}{B} \cdot \frac{B}{2} \cdot \frac{6}{7}L + \frac{LN}{7680} \leq \frac{7LN}{8}.
\end{aligned} \tag{B.142}$$

We next bound the terms in (B.141) that involve summation of square roots. From $N/B \geq 2$, we have $\lfloor \frac{N}{2B} \rfloor \geq \frac{N}{3B}$ and $\lfloor \frac{N}{2B} \rfloor + 1 \geq \frac{N}{2B}$, so

$$\begin{aligned}
& \sum_{i=0}^{\lfloor \frac{N}{2B} \rfloor} \sum_{j=\frac{B}{4}}^{\frac{B}{2}} \left(\sqrt{\frac{iB}{M}} + \sqrt{j} \right) = \left(\frac{B}{4} + 1 \right) \sqrt{\frac{B}{M}} \sum_{i=0}^{\lfloor \frac{N}{2B} \rfloor} \sqrt{i} + \left(\left\lfloor \frac{N}{2B} \right\rfloor + 1 \right) \sum_{j=\frac{B}{4}}^{\frac{B}{2}} \sqrt{j} \\
& \geq \frac{B^{3/2}}{4M^{1/2}} \int_0^{\lfloor \frac{N}{2B} \rfloor} \sqrt{t} dt + \frac{N}{2B} \int_{\frac{B}{4}-1}^{\frac{B}{2}} \sqrt{t} dt \\
& \geq \frac{B^{3/2}}{6M^{1/2}} \left(\frac{N}{3B} \right)^{3/2} + \frac{N}{3B} \left[\left(\frac{B}{2} \right)^{3/2} - \left(\frac{B}{4} \right)^{3/2} \right] \\
& = \frac{N^{3/2}}{18\sqrt{3}M^{1/2}} + \frac{(2\sqrt{2} - 1)NB^{1/2}}{24}.
\end{aligned} \tag{B.143}$$

For the other sum, we have

$$\begin{aligned}
& \sum_{i=0}^{\frac{N}{B}-1} \sum_{j=0}^{B-1} \left(\sqrt{\frac{iB}{M}} + \sqrt{j} \right) = \frac{B^{3/2}}{M^{1/2}} \sum_{i=0}^{\frac{N}{B}-1} \sqrt{i} + \frac{N}{B} \sum_{j=0}^{B-1} \sqrt{j} \\
& \leq \frac{B^{3/2}}{M^{1/2}} \int_0^{\frac{N}{B}} \sqrt{t} dt + \frac{N}{B} \int_0^B \sqrt{t} dt \\
& = \frac{2B^{3/2}}{3M^{1/2}} \left(\frac{N}{B} \right)^{3/2} + \frac{2N}{3B} B^{3/2} \\
& = \frac{2N^{3/2}}{3M^{1/2}} + \frac{2NB^{1/2}}{3}.
\end{aligned} \tag{B.144}$$

Substituting the bounds (B.142), (B.143), and (B.144) into (B.141), and using $\mu \leq \frac{L}{40000}$,

$$\mathbb{E}[y_{\frac{N}{B}} - y_0] \geq -\frac{7\eta LN}{8} y_0 + \frac{\eta^2 L \nu}{1536} \left(\frac{N^{3/2}}{18\sqrt{3}M^{1/2}} + \frac{(2\sqrt{2} - 1)NB^{1/2}}{24} \right)$$

$$\begin{aligned}
& -\frac{1025\eta^2\mu\nu}{1024} \left(\frac{2N^{3/2}}{3M^{1/2}} + \frac{2NB^{1/2}}{3} \right) \\
& \geq -\frac{7\eta LN}{8}y_0 + \frac{\eta^2 L\nu}{240000} \left(\frac{N^{3/2}}{M^{1/2}} + NB^{1/2} \right). \tag{B.145}
\end{aligned}$$

For the other case $y_0 < 0$, we have the following lemma, a local RR counterpart of Lemma B.6. In Appendix B.8.3, we prove the following:

Lemma B.11. *If $\eta \leq \frac{1}{1025LN}$ and an epoch starts at $y_0 < 0$, then*

$$\mathbb{E}\left[y_{\frac{N}{B}} \mid y_0 < 0\right] \geq \left(1 - \frac{7\eta LN}{8}\right)y_0.$$

Further, if the first epoch of the algorithm is initialized at 0, then for any starting iterate y_0 of any following epoch, we have $\mathbb{P}(y_0 \geq 0) \geq 1/2$.

Using (B.145) and Lemma B.11, we get

$$\begin{aligned}
\mathbb{E}[y_{\frac{N}{B}}] &= \mathbb{P}(y_0 \geq 0)\mathbb{E}[y_{\frac{N}{B}} \mid y_0 \geq 0] + \mathbb{P}(y_0 < 0)\mathbb{E}[y_{\frac{N}{B}} \mid y_0 < 0] \\
&\geq \mathbb{P}(y_0 \geq 0) \left(\left(1 - \frac{7\eta LN}{8}\right)y_0 + \frac{\eta^2 L\nu}{240000} \left(\frac{N^{3/2}}{M^{1/2}} + NB^{1/2} \right) \right) \\
&\quad + \mathbb{P}(y_0 < 0) \left(1 - \frac{7\eta LN}{8}\right)y_0 \\
&\geq \left(1 - \frac{7\eta LN}{8}\right)y_0 + \frac{\eta^2 L\nu}{480000} \left(\frac{N^{3/2}}{M^{1/2}} + NB^{1/2} \right).
\end{aligned}$$

Thus far, we have characterized the expected per-epoch update, starting from the initial iterate y_0 and iterating until the last iterate $y_{\frac{N}{B}}$ of the epoch. Now recall that we run the algorithm for K epochs. Using $y_{k,i}$ to denote the i -th aggregated iterate of the k -th epoch, we get a lower bound on the expectation of the last iterate $y_{k,\frac{N}{B}}$ if we initialize at $y_{1,0} = 0$:

$$\begin{aligned}
\mathbb{E}[y_{K,\frac{N}{B}}] &\geq \left(1 - \frac{7\eta LN}{8}\right)^K y_{1,0} + \frac{\eta^2 L\nu}{480000} \left(\frac{N^{3/2}}{M^{1/2}} + NB^{1/2} \right) \sum_{k=0}^{K-1} \left(1 - \frac{7\eta LN}{8}\right)^k \\
&= \frac{\eta^2 L\nu}{480000} \left(\frac{N^{3/2}}{M^{1/2}} + NB^{1/2} \right) \frac{1 - \left(1 - \frac{7\eta LN}{8}\right)^K}{\frac{7\eta LN}{8}} \\
&= \frac{\eta\nu}{420000} \left(\frac{N^{1/2}}{M^{1/2}} + B^{1/2} \right) \left(1 - \left(1 - \frac{7\eta LN}{8}\right)^K\right) \\
&\geq \frac{\eta\nu}{420000} \left(\frac{N^{1/2}}{M^{1/2}} + B^{1/2} \right) \left(1 - \left(1 - \frac{7L}{8\mu K}\right)^K\right). \tag{Since $\eta \geq \frac{1}{\mu NK}$ }
\end{aligned}$$

Note that since $\frac{L}{\mu} \geq 40000$ and $K \geq \frac{1025L}{\mu}$ (which is implied by $\frac{1}{\mu NK} \leq \eta \leq \frac{1}{1025LN}$),

$$1 - \left(1 - \frac{7L}{8\mu K}\right)^K \geq 1 - e^{-\frac{7L}{8\mu}} \geq 1 - e^{-35000} \approx 1.$$

Hence, we get from $\eta \geq \frac{1}{\mu NK}$ that

$$\mathbb{E}[y_{K, \frac{N}{B}}] = \Omega\left(\frac{\eta \nu N^{1/2}}{M^{1/2}} + \eta \nu B^{1/2}\right) = \Omega\left(\frac{\nu}{\mu M^{1/2} N^{1/2} K} + \frac{\nu B^{1/2}}{\mu N K}\right),$$

and by Jensen's inequality, we finally have

$$\mathbb{E}[F(y_{K, \frac{N}{B}})] \geq \frac{1}{2} \mathbb{E}[\mu y_{K, \frac{N}{B}}^2] = \Omega(\mu \mathbb{E}[y_{K, \frac{N}{B}}]^2) = \Omega\left(\frac{\nu^2}{\mu M N K^2} + \frac{\nu^2 B}{\mu N^2 K^2}\right).$$

Recall that, from the paragraph below Lemmas B.9 and B.10 to this point, we have assumed $N/B \geq 2$. We handle the case $B = N$ now. In this case, notice that all the $\sqrt{\frac{iB}{M}}$ terms that appear in (B.141) disappear, because we always have $i = 0$. Therefore, the proof goes through in the same way, modulo the fact that we do not have the terms that originate from the $\sqrt{\frac{iB}{M}}$ terms in (B.141). Therefore, we can show

$$\mathbb{E}[F(y_{K, \frac{N}{B}})] \geq \frac{1}{2} \mathbb{E}[\mu y_{K, \frac{N}{B}}^2] = \Omega(\mu \mathbb{E}[y_{K, \frac{N}{B}}]^2) = \Omega\left(\frac{\nu^2}{\mu N K^2}\right).$$

B.7.3 Lower bound for $\eta \geq \frac{1}{\mu NK}$ and $\eta \geq \frac{1}{1025LN}$

Similar to earlier parts of the proof, here as well, each machine will have the same component functions, that is, there will be no inter-machine deviation. The proof uses a similar construction as Safran and Shamir (2020, 2021):

$$F_3(x) := \frac{1}{N} \left(\sum_{i=1}^{\frac{N}{2}} f_{+1}(x) + \sum_{i=\frac{N}{2}+1}^N f_{-1}(x) \right), \text{ where}$$

$$f_{+1}(x) := \frac{Lx^2}{2} + \nu x, \text{ and } f_{-1}(x) := \frac{Lx^2}{2} - \nu x.$$

Hence, $F_3(x) = \frac{Lx^2}{2}$, and has its minimizer at 0.

We first compute the expected “progress” over a given epoch. For simplicity, let us omit the subscript for epochs for now. Let y_0 denote the iterate at the beginning of the epoch (which is the same across all the machines $x_0^m = y_0$) and x_i^m denote the i -th local iterate for machine m . After every B local iterates, the server aggregates the local iterates x_{iB}^m ,

computes their average $y_i := \frac{1}{M} \sum_{m=1}^M x_{iB}^m$, and synchronizes all the machines $x_{iB}^m := y_i$.

For the epoch, let σ^m be the permutation of $\frac{N}{2} + 1$'s and $\frac{N}{2} - 1$'s sampled by machine m . Upon receiving the aggregated iterate $x_{iB}^m = y_i$, each machine m performs B local updates. Unrolling the local update rules, the iterate after the B updates (and before synchronization) can be written as follows:

$$\begin{aligned}
x_{(i+1)B}^m &= x_{(i+1)B-1}^m - \eta \nabla f_{\sigma_{(i+1)B}^m}(x_{(i+1)B-1}^m) \\
&= x_{(i+1)B-1}^m - \eta(Lx_{(i+1)B-1}^m + \nu\sigma_{(i+1)B}^m) \\
&= (1 - \eta L)x_{(i+1)B-1}^m - \eta\nu\sigma_{(i+1)B}^m \\
&= \dots \\
&= (1 - \eta L)^B x_{iB}^m - \eta\nu \sum_{j=iB+1}^{(i+1)B} (1 - \eta L)^{(i+1)B-j} \sigma_j^m \\
&= (1 - \eta L)^B y_i - \eta\nu \sum_{j=iB+1}^{(i+1)B} (1 - \eta L)^{(i+1)B-j} \sigma_j^m,
\end{aligned}$$

After synchronization, we get

$$y_{i+1} := \frac{1}{M} \sum_{m=1}^M x_{(i+1)B}^m = (1 - \eta L)^B y_i - \frac{\eta\nu}{M} \sum_{m=1}^M \sum_{j=iB+1}^{(i+1)B} (1 - \eta L)^{(i+1)B-j} \sigma_j^m.$$

Unrolling the equation above from $y_{\frac{N}{B}}$ (the final iterate of the epoch, after synchronization) to y_0 (the starting iterate), we get

$$\begin{aligned}
y_{\frac{N}{B}} &= (1 - \eta L)^B y_{\frac{N}{B}-1} - \frac{\eta\nu}{M} \sum_{m=1}^M \sum_{j=N-B+1}^N (1 - \eta L)^{N-j} \sigma_j^m \\
&= \dots \\
&= (1 - \eta L)^N y_0 - \sum_{i=1}^{\frac{N}{B}} (1 - \eta L)^{N-iB} \frac{\eta\nu}{M} \sum_{m=1}^M \sum_{j=(i-1)B+1}^{iB} (1 - \eta L)^{iB-j} \sigma_j^m \\
&= (1 - \eta L)^N y_0 - \frac{\eta\nu}{M} \sum_{m=1}^M \sum_{j=1}^N (1 - \eta L)^{N-j} \sigma_j^m.
\end{aligned}$$

Then, by squaring both sides and taking expectations,

$$\mathbb{E}[y_{\frac{N}{B}}^2] = (1 - \eta L)^{2N} y_0^2 - \frac{2\eta\nu(1 - \eta L)^N x_0}{M} \mathbb{E} \left[\sum_{m=1}^M \sum_{i=1}^N (1 - \eta L)^{N-i} \sigma_i^m \right]$$

$$\begin{aligned}
& + \frac{\eta^2 \nu^2}{M^2} \mathbb{E} \left[\left(\sum_{m=1}^M \sum_{i=1}^N (1 - \eta L)^{N-i} \sigma_i^m \right)^2 \right] \\
& = (1 - \eta L)^{2N} y_0^2 + \frac{\eta^2 \nu^2}{M^2} \mathbb{E} \left[\left(\sum_{m=1}^M \sum_{i=1}^N (1 - \eta L)^{N-i} \sigma_i^m \right)^2 \right], \tag{B.146}
\end{aligned}$$

where we used the fact that $\mathbb{E}[\sigma_i^m] = 0$. Further, because σ^m and $\sigma^{m'}$ are independent' and identically distributed for different m and m' , we get that

$$\begin{aligned}
& \mathbb{E} \left[\left(\frac{\eta \nu}{M} \sum_{m=1}^M \sum_{i=1}^N (1 - \eta L)^{N-i} \sigma_i^m \right)^2 \right] \\
& = \sum_{m=1}^M \mathbb{E} \left[\left(\sum_{i=1}^N (1 - \eta L)^{N-i} \sigma_i^m \right)^2 \right] \\
& \quad + \sum_{m \neq m'} \mathbb{E} \left[\sum_{i=1}^N (1 - \eta L)^{N-i} \sigma_i^m \right] \mathbb{E} \left[\sum_{i=1}^N (1 - \eta L)^{N-i} \sigma_i^{m'} \right] \\
& = M \mathbb{E} \left[\left(\sum_{i=1}^N (1 - \eta L)^{N-i} \sigma_i^1 \right)^2 \right],
\end{aligned}$$

where the last equality used the fact that $\mathbb{E}[\sigma_j^m] = 0$ for all $m \in [M]$ and $i \in [N]$, and that σ^m are identically distributed. Since we only consider the permutation σ^1 (i.e., the one for machine 1) from now on, we henceforth omit the superscript. Substituting this to (B.146) gives

$$\mathbb{E}[y_{\frac{N}{B}}^2] = (1 - \eta L)^{2N} y_0^2 + \underbrace{\frac{\eta^2 \nu^2}{M} \mathbb{E} \left[\left(\sum_{i=1}^N (1 - \eta L)^{N-i} \sigma_i \right)^2 \right]}_{:=\Phi}. \tag{B.147}$$

From (B.147), we have calculated the per-epoch expected update, because the final iterate $y_{\frac{N}{B}}$ is also the initial iterate of the next epoch. Recall that we run the algorithm for K epochs. We now use $y_{k,i}$ to denote the iterate after the i -th communication round in the k -th epoch. Using (B.147), we get a lower bound on the expectation of the last iterate $y_{K, \frac{N}{B}}$ squared:

$$\mathbb{E}[y_{K, \frac{N}{B}}^2] = (1 - \eta L)^{2NK} y_{1,0}^2 + \frac{\eta^2 \nu^2}{M} \Phi \sum_{k=0}^{K-1} (1 - \eta L)^{2Nk} \geq \frac{\eta^2 \nu^2}{M} \Phi. \tag{B.148}$$

where the inequality used that we initialize at $y_{1,0} = 0$ and $\sum_{k=0}^{K-1} (1 - \eta L)^{2Nk} \geq (1 - \eta L)^0 = 1$. Next, we bound the expectation term, i.e., Φ , defined in (B.147). Using Lemma 1 from

Safran and Shamir (2020) with n and α replaced with N and ηL respectively, we have

$$\Phi \geq c \cdot \min \left\{ \frac{1}{\eta L}, \eta^2 L^2 N^3 \right\}, \quad (\text{B.149})$$

for some universal constant $c > 0$. Using the fact that $\eta \geq \frac{1}{1025LN}$, it is easy to check that the RHS of (B.149) is lower-bounded by $\frac{c'}{\eta L}$, where $c' > 0$ is a universal constant. Combining (B.148) and (B.149) gives

$$\mathbb{E}[y_{K, \frac{N}{B}}^2] \geq \frac{\eta^2 \nu^2}{M} \cdot \frac{c'}{\eta L} = \frac{c' \eta \nu^2}{LM}.$$

Also using the fact that $\eta \geq \frac{1}{\mu N K}$, we get

$$\mathbb{E}[F_3(y_{K, \frac{N}{B}})] = \frac{L}{2} \mathbb{E}[y_{K, \frac{N}{B}}^2] \geq \frac{c' \nu^2}{2\mu M N K}.$$

B.8 Proofs of helper lemmas for Appendix B.7

B.8.1 Proof of Lemma B.9

The proof of Lemma B.9 is similar to its minibatch RR counterpart, Lemma B.4. From the given $0 \leq i \leq \frac{N}{B} - 1$, $0 \leq j \leq B - 1$, define $k := iB + j$, in order to simplify notation. We also define $\mathcal{E} := \frac{1}{M} \left(\sum_{m=1}^M \sum_{l=1}^{iB} \sigma_l^m \right) + \sum_{l=iB+1}^k \sigma_l^1$.

By the law of total expectation we have

$$\begin{aligned} \mathbb{E}[(L1_{x_k^1 \leq 0} + \mu 1_{x_k^1 > 0})x_k^1] &= \mathbb{P}(\mathcal{E} > 0) \mathbb{E}[(L1_{x_k^1 \leq 0} + \mu 1_{x_k^1 > 0})x_k^1 | \mathcal{E} > 0] \\ &\quad + \mathbb{P}(\mathcal{E} \leq 0) \mathbb{E}[(L1_{x_k^1 \leq 0} + \mu 1_{x_k^1 > 0})x_k^1 | \mathcal{E} \leq 0] \\ &\leq \mathbb{P}(\mathcal{E} > 0) L \mathbb{E}[x_k^1 | \mathcal{E} > 0] + \mathbb{P}(\mathcal{E} \leq 0) \mu \mathbb{E}[x_k^1 | \mathcal{E} \leq 0], \end{aligned} \quad (\text{B.150})$$

where the last inequality used the fact that $(L1_{t \leq 0} + \mu 1_{t > 0})t \leq Lt$ and $(L1_{t \leq 0} + \mu 1_{t > 0})t \leq \mu t$ for any $t \in \mathbb{R}$.

We handle each of the two expectations in (B.150) separately. We first bound $\mathbb{E}[x_k^1 | \mathcal{E} > 0]$. Recall from the definition of algorithm iterates that

$$x_k^1 - y_0 = x_{iB+j}^1 - y_0 = -\frac{\eta}{M} \sum_{m=1}^M \sum_{l=0}^{iB-1} \nabla f_{\sigma_{l+1}^m}(x_l^m) - \eta \sum_{l=iB}^{k-1} f_{\sigma_{l+1}^1}(x_l^1). \quad (\text{B.151})$$

Expanding (B.151) using the definition of $\nabla f_{\sigma_{l+1}^m}$'s, we obtain

$$\mathbb{E}[x_k^1 | \mathcal{E} > 0]$$

$$\begin{aligned}
&= \mathbb{E} \left[y_0 - \frac{\eta}{M} \sum_{m=1}^M \sum_{l=0}^{iB-1} \left(\nu \sigma_{l+1}^m + (L1_{x_l^m \leq 0} + \mu 1_{x_l^m > 0}) x_l^m \right) \right. \\
&\quad \left. - \eta \sum_{l=iB}^{k-1} \left(\nu \sigma_{l+1}^1 + (L1_{x_l^1 \leq 0} + \mu 1_{x_l^1 > 0}) x_l^1 \right) \middle| \mathcal{E} > 0 \right] \\
&= \mathbb{E} \left[y_0 - \frac{\eta}{M} \sum_{m=1}^M \sum_{l=0}^{iB-1} (L1_{x_l^m \leq 0} + \mu 1_{x_l^m > 0}) (x_l^m - y_0) \right. \\
&\quad \left. - \eta \sum_{l=iB}^{k-1} (L1_{x_l^1 \leq 0} + \mu 1_{x_l^1 > 0}) (x_l^1 - y_0) \middle| \mathcal{E} > 0 \right] - \eta \nu \mathbb{E} [\mathcal{E} | \mathcal{E} > 0] \\
&\quad - \mathbb{E} \left[\frac{\eta}{M} \sum_{m=1}^M \sum_{l=0}^{iB-1} (L1_{x_l^m \leq 0} + \mu 1_{x_l^m > 0}) y_0 + \eta \sum_{l=iB}^{k-1} (L1_{x_l^1 \leq 0} + \mu 1_{x_l^1 > 0}) y_0 \middle| \mathcal{E} > 0 \right] \\
&= y_0 \mathbb{E} \left[1 - \frac{\eta}{M} \sum_{m=1}^M \sum_{l=0}^{iB-1} (L1_{x_l^m \leq 0} + \mu 1_{x_l^m > 0}) - \eta \sum_{l=iB}^{k-1} (L1_{x_l^1 \leq 0} + \mu 1_{x_l^1 > 0}) \middle| \mathcal{E} > 0 \right] \\
&\quad - \mathbb{E} \left[\frac{\eta}{M} \sum_{m=1}^M \sum_{l=0}^{iB-1} (L1_{x_l^m \leq 0} + \mu 1_{x_l^m > 0}) (x_l^m - y_0) \right. \\
&\quad \left. + \eta \sum_{l=iB}^{k-1} (L1_{x_l^1 \leq 0} + \mu 1_{x_l^1 > 0}) (x_l^1 - y_0) \middle| \mathcal{E} > 0 \right] - \eta \nu \mathbb{E} [\mathcal{E} | \mathcal{E} > 0] \\
&\leq y_0 \mathbb{E} \left[1 - \frac{\eta}{M} \sum_{m=1}^M \sum_{l=0}^{iB-1} (L1_{x_l^m \leq 0} + \mu 1_{x_l^m > 0}) - \eta \sum_{l=iB}^{k-1} (L1_{x_l^1 \leq 0} + \mu 1_{x_l^1 > 0}) \middle| \mathcal{E} > 0 \right] \\
&\quad + \frac{\eta L(M-1)}{M} \sum_{l=0}^{iB-1} \mathbb{E} [|x_l^2 - y_0| | \mathcal{E} > 0] + \frac{\eta L}{M} \sum_{l=0}^{iB-1} \mathbb{E} [|x_l^1 - y_0| | \mathcal{E} > 0] \\
&\quad + \eta L \sum_{l=iB}^{k-1} \mathbb{E} [|x_l^1 - y_0| | \mathcal{E} > 0] - \eta \nu \mathbb{E} [\mathcal{E} | \mathcal{E} > 0],
\end{aligned}$$

where the last inequality used the fact that $(L1_{t \leq 0} + \mu 1_{t > 0})t \leq Lt$ and $(L1_{t \leq 0} + \mu 1_{t > 0})t \leq \mu t$ for any $t \in \mathbb{R}$; and that for different $m = 2, \dots, M$, the local iterates x_l^m are identically distributed conditioned on $\mathcal{E} > 0$. Next, we use the fact that for any nonnegative random variable v and event Δ , we have that $\mathbb{E}[v | \Delta] \leq \mathbb{E}[v] / \mathbb{P}(\Delta)$. Hence,

$$\begin{aligned}
&\mathbb{E} [x_k^1 | \mathcal{E} > 0] \\
&\leq y_0 \mathbb{E} \left[1 - \frac{\eta}{M} \sum_{m=1}^M \sum_{l=0}^{iB-1} (L1_{x_l^m \leq 0} + \mu 1_{x_l^m > 0}) - \eta \sum_{l=iB}^{k-1} (L1_{x_l^1 \leq 0} + \mu 1_{x_l^1 > 0}) \middle| \mathcal{E} > 0 \right] \\
&\quad + \frac{\eta L(M-1)}{M} \sum_{l=0}^{iB-1} \frac{\mathbb{E} [|x_l^2 - y_0|]}{\mathbb{P}(\mathcal{E} > 0)} + \frac{\eta L}{M} \sum_{l=0}^{iB-1} \frac{\mathbb{E} [|x_l^1 - y_0|]}{\mathbb{P}(\mathcal{E} > 0)} + \eta L \sum_{l=iB}^{k-1} \frac{\mathbb{E} [|x_l^1 - y_0|]}{\mathbb{P}(\mathcal{E} > 0)} \\
&\quad - \eta \nu \mathbb{E} [\mathcal{E} | \mathcal{E} > 0] \\
&= y_0 \mathbb{E} \left[1 - \frac{\eta}{M} \sum_{m=1}^M \sum_{l=0}^{iB-1} (L1_{x_l^m \leq 0} + \mu 1_{x_l^m > 0}) - \eta \sum_{l=iB}^{k-1} (L1_{x_l^1 \leq 0} + \mu 1_{x_l^1 > 0}) \middle| \mathcal{E} > 0 \right]
\end{aligned}$$

$$- \eta \nu \mathbb{E} [\mathcal{E} | \mathcal{E} > 0] + \eta L \sum_{l=0}^{k-1} \frac{\mathbb{E} [|x_l^1 - y_0|]}{\mathbb{P}(\mathcal{E} > 0)} \quad (\text{B.152})$$

where the last equality used the fact that for different $m \in [M]$, the local iterates x_l^m are identically distributed when they are not conditioned.

Next, we use Lemma B.7 again to bound the conditional expectations that arise in (B.152). We restate the lemma for the reader's convenience.

Lemma B.12. *For $m \in [M]$, let σ^m be a random permutation of $\frac{N}{2} + 1$'s and $\frac{N}{2} - 1$'s. Then, for any $i \leq \frac{N}{2}$ and $k \leq \frac{B}{2}$, we have*

$$\frac{1}{64} \left(\sqrt{\frac{i}{M}} + \sqrt{k} \right) \leq \mathbb{E} \left[\left| \left(\frac{1}{M} \sum_{m=1}^M \sum_{j=1}^i \sigma_j^m \right) + \sum_{j=i+1}^{i+k} \sigma_j^M \right| \right].$$

Furthermore, for any $0 \leq i \leq N$ and $0 \leq k \leq N$ satisfying $i + k \leq N$, we have

$$\mathbb{E} \left[\left| \left(\frac{1}{M} \sum_{m=1}^M \sum_{j=1}^i \sigma_j^m \right) + \sum_{j=i+1}^{i+k} \sigma_j^M \right| \right] \leq \sqrt{\frac{i}{M}} + \sqrt{k}.$$

Lastly, for any $0 \leq i \leq \frac{N}{2}$ and $0 \leq k \leq \frac{B}{2}$ satisfying $i + k \geq 1$, we have

$$\mathbb{P} \left(\sum_{m=1}^M \sum_{j=1}^i \sigma_j^m + M \sum_{j=i+1}^{i+k} \sigma_j^M > 0 \right) = \mathbb{P} \left(\sum_{m=1}^M \sum_{j=1}^i \sigma_j^m + M \sum_{j=i+1}^{i+k} \sigma_j^M < 0 \right) \geq \frac{1}{6}.$$

Lemma B.7 implies that $\mathbb{E} [\mathcal{E} | \mathcal{E} > 0] \in \left[\frac{1}{64} \left(\sqrt{\frac{iB}{M}} + \sqrt{j} \right), \sqrt{\frac{iB}{M}} + \sqrt{j} \right]$ and $\mathbb{P}(\mathcal{E} > 0) = \mathbb{P}(\mathcal{E} < 0) \geq 1/6$. From this, we get

$$\eta \nu \mathbb{E} [\mathcal{E} | \mathcal{E} > 0] \geq \frac{\eta \nu}{64} \left(\sqrt{\frac{iB}{M}} + \sqrt{j} \right), \quad (\text{B.153})$$

$$\frac{\mathbb{E} [|x_l^1 - y_0|]}{\mathbb{P}(\mathcal{E} > 0)} \leq 6 \mathbb{E} [|x_l^1 - y_0|]. \quad (\text{B.154})$$

Also, since $\eta \leq \frac{1}{LN}$ we have

$$1 - k\eta\mu \geq 1 - \frac{\eta}{M} \sum_{m=1}^M \sum_{l=0}^{iB-1} (L1_{x_l^m \leq 0} + \mu 1_{x_l^m > 0}) - \eta \sum_{l=iB}^{k-1} (L1_{x_l^1 \leq 0} + \mu 1_{x_l^1 > 0}) \geq 1 - \eta LN \geq 0,$$

which implies that

$$y_0 \mathbb{E} \left[1 - \frac{\eta}{M} \sum_{m=1}^M \sum_{l=0}^{iB-1} (L1_{x_l^m \leq 0} + \mu 1_{x_l^m > 0}) - \eta \sum_{l=iB}^{k-1} (L1_{x_l^1 \leq 0} + \mu 1_{x_l^1 > 0}) \middle| \mathcal{E} > 0 \right] \leq (1 - k\eta\mu)y_0. \quad (\text{B.155})$$

Substituting (B.153), (B.154), and (B.155) to (B.152), we obtain

$$\mathbb{E} [x_k^1 | \mathcal{E} > 0] \leq (1 - k\eta\mu)y_0 - \frac{\eta\nu}{64} \left(\sqrt{\frac{iB}{M}} + \sqrt{j} \right) + 6\eta L \sum_{l=0}^{k-1} \mathbb{E} [|x_l^1 - y_0|]. \quad (\text{B.156})$$

Next, we have the following lemma that we can apply to $\mathbb{E}[|x_l^1 - y_0|]$. Proof of Lemma B.13 can be found in Appendix B.8.4.

Lemma B.13. For $y_0 \geq 0$, $0 \leq i \leq \frac{N}{B} - 1$, $0 \leq j \leq B - 1$, and $\eta \leq \frac{1}{1025LN}$,

$$\mathbb{E}[|x_{iB+j}^1 - y_0|] \leq \frac{1025}{1024} \eta\nu \left(\sqrt{\frac{iB}{M}} + \sqrt{j} \right) + \frac{1025}{1024} (iB + j) \eta L y_0.$$

Applying this lemma to (B.156) and arranging the bounds (recall that $k := iB + j$), we get

$$\begin{aligned} & \frac{1024}{1025} \sum_{l=0}^{k-1} \mathbb{E} [|x_l^1 - y_0|] \\ & \leq \eta\nu \sum_{l=0}^{k-1} \left(\sqrt{\frac{B[l/B]}{M}} + \sqrt{l - B[l/B]} \right) + \eta L y_0 \sum_{l=0}^{k-1} l \\ & = \eta\nu \sum_{l=0}^{iB-1} \left(\sqrt{\frac{B[l/B]}{M}} + \sqrt{l - B[l/B]} \right) \\ & \quad + \eta\nu \sum_{l=iB}^{iB+j-1} \left(\sqrt{\frac{B[l/B]}{M}} + \sqrt{l - B[l/B]} \right) + \eta L y_0 \sum_{l=0}^{k-1} l \\ & = \eta\nu \left(B \sum_{l=0}^{i-1} \sqrt{\frac{lB}{M}} + j \sqrt{\frac{iB}{M}} + i \sum_{l=0}^{B-1} \sqrt{l} + \sum_{l=0}^{j-1} \sqrt{l} \right) + \eta L y_0 \sum_{l=0}^{k-1} l. \end{aligned} \quad (\text{B.157})$$

The terms in (B.157) can be bounded using $\sum_{l=0}^{c-1} \sqrt{l} \leq \int_0^c \sqrt{t} dt$:

$$\frac{1024}{1025} \sum_{l=0}^{k-1} \mathbb{E} [|x_l^1 - y_0|] \leq \eta\nu \left(\frac{2i^{3/2}B^{3/2}}{3M^{1/2}} + \frac{i^{1/2}jB^{1/2}}{M^{1/2}} + \frac{2iB^{3/2}}{3} + \frac{2j^{3/2}}{3} \right) + \frac{k^2\eta L y_0}{2}$$

$$\begin{aligned}
&\leq \eta\nu \left(\frac{(2iB + 3j)}{3} \sqrt{\frac{iB}{M}} + \frac{4iBj^{1/2}}{3} + \frac{2j^{3/2}}{3} \right) + \frac{k^2\eta Ly_0}{2} \\
&= \eta\nu \left(\frac{(2iB + 3j)}{3} \sqrt{\frac{iB}{M}} + \frac{(4iB + 2j)}{3} \sqrt{j} \right) + \frac{k^2\eta Ly_0}{2} \\
&\leq \frac{4k\eta\nu}{3} \left(\sqrt{\frac{iB}{M}} + \sqrt{j} \right) + \frac{k^2\eta Ly_0}{2}, \tag{B.158}
\end{aligned}$$

where the second last inequality used $\frac{B}{4} \leq j$ (and hence $B^{1/2} \leq 2j^{1/2}$), and the last inequality used $2iB + 3j \leq 4(iB + j) = 4k$ and $4iB + 2j \leq 4(iB + j) = 4k$. Substituting (B.158) to (B.156), we get

$$\begin{aligned}
&\mathbb{E} [x_k^1 | \mathcal{E} > 0] \tag{B.159} \\
&\leq (1 - k\eta\mu)y_0 - \frac{\eta\nu}{64} \left(\sqrt{\frac{iB}{M}} + \sqrt{j} \right) + \frac{1025k\eta^2 L\nu}{128} \left(\sqrt{\frac{iB}{M}} + \sqrt{j} \right) + \frac{3075k^2\eta^2 L^2 y_0}{1024} \\
&= \left(1 - k\eta\mu + \frac{3075k^2\eta^2 L^2}{1024} \right) y_0 - \left(\frac{1}{64} - \frac{1025k\eta L}{128} \right) \eta\nu \left(\sqrt{\frac{iB}{M}} + \sqrt{j} \right) \\
&\leq \left(1 - k\eta\mu + \frac{3k\eta L}{1024} \right) y_0 - \frac{\eta\nu}{128} \left(\sqrt{\frac{iB}{M}} + \sqrt{j} \right). \tag{B.160}
\end{aligned}$$

The last inequality here used $k\eta L \leq \eta L N \leq \frac{1}{1025}$, which follows from $\eta \leq \frac{1}{1025LN}$. Thus far, we have obtained an upper bound for $\mathbb{E} [x_k^1 | \mathcal{E} > 0]$.

Recall that there is another conditional expectation in (B.150) that we want to bound, namely $\mathbb{E} [x_k^1 | \mathcal{E} \leq 0]$. We bound it below, using the tools developed so far. For $i \leq \frac{N}{2B}$ and $\frac{B}{4} \leq j \leq \frac{B}{2}$,

$$\begin{aligned}
\mathbb{E} [x_k^1 | \mathcal{E} \leq 0] &= y_0 + \mathbb{E} [x_k^1 - y_0 | \mathcal{E} \leq 0] \\
&\leq y_0 + \mathbb{E} [|x_k^1 - y_0| | \mathcal{E} \leq 0] \\
&\leq y_0 + \frac{\mathbb{E} [|x_k^1 - x_0|]}{\mathbb{P}(\mathcal{E} \leq 0)} \\
&\leq y_0 + 6\mathbb{E} [|x_k^1 - x_0|] \tag{Using Lemma B.7} \\
&\leq y_0 + \frac{3075\eta\nu}{512} \left(\sqrt{\frac{iB}{M}} + \sqrt{j} \right) + \frac{3075k\eta Ly_0}{512} \tag{Using Lemma B.13} \\
&\leq \left(1 + \frac{3075k\eta L}{512} \right) y_0 + \frac{3075\eta\nu}{512} \left(\sqrt{\frac{iB}{M}} + \sqrt{j} \right). \tag{B.161}
\end{aligned}$$

Using (B.160) and (B.161) in (B.150), we get that for $i \leq \frac{N}{2B}$ and $\frac{B}{4} \leq j \leq \frac{B}{2}$:

$$\begin{aligned}
& \mathbb{E}[(L1_{x_k^1 \leq 0} + \mu 1_{x_k^1 > 0})x_k^1] \\
& \leq \mathbb{P}(\mathcal{E} > 0) L \mathbb{E}[x_k^1 | \mathcal{E} > 0] + \mathbb{P}(\mathcal{E} \leq 0) \mu \mathbb{E}[x_k^1 | \mathcal{E} \leq 0] \\
& \leq \mathbb{P}(\mathcal{E} > 0) L \left(\left(1 - k\eta\mu + \frac{3k\eta L}{1024}\right) y_0 - \frac{\eta\nu}{128} \left(\sqrt{\frac{iB}{M}} + \sqrt{j}\right) \right) \\
& \quad + \mathbb{P}(\mathcal{E} \leq 0) \mu \left(\left(1 + \frac{3075k\eta L}{512}\right) y_0 + \frac{3075\eta\nu}{512} \left(\sqrt{\frac{iB}{M}} + \sqrt{j}\right) \right). \tag{B.162}
\end{aligned}$$

From Lemma B.7, note that $\frac{1}{6} \leq \mathbb{P}(\mathcal{E} > 0) \leq \frac{5}{6}$ and $\frac{1}{6} \leq \mathbb{P}(\mathcal{E} \leq 0) \leq \frac{5}{6}$. We use these inequalities, along with $k\eta L \leq \eta L N \leq \frac{1}{1025}$ and $\frac{L}{\mu} \geq \frac{15375}{2}$, to bound the terms appearing in (B.162).

$$\begin{aligned}
& \mathbb{P}(\mathcal{E} > 0) L \left(1 - k\eta\mu + \frac{3k\eta L}{1024}\right) y_0 + \mathbb{P}(\mathcal{E} \leq 0) \mu \left(1 + \frac{3075k\eta L}{512}\right) y_0 \\
& \leq \frac{5}{6} L \left(1 + \frac{3}{1049600}\right) y_0 + \frac{5}{6} \cdot \frac{2L}{15375} \left(1 + \frac{3}{512}\right) y_0 \leq \frac{6}{7} L y_0. \tag{B.163}
\end{aligned}$$

We also have

$$\begin{aligned}
& -\mathbb{P}(\mathcal{E} > 0) \frac{\eta L \nu}{128} \left(\sqrt{\frac{iB}{M}} + \sqrt{j}\right) + \mathbb{P}(\mathcal{E} \leq 0) \frac{3075\eta\mu\nu}{512} \left(\sqrt{\frac{iB}{M}} + \sqrt{j}\right) \\
& \leq -\frac{\eta L \nu}{768} \left(\sqrt{\frac{iB}{M}} + \sqrt{j}\right) + \frac{5125\eta\mu\nu}{1024} \left(\sqrt{\frac{iB}{M}} + \sqrt{j}\right) \\
& \leq -\frac{\eta L \nu}{1536} \left(\sqrt{\frac{iB}{M}} + \sqrt{j}\right), \tag{B.164}
\end{aligned}$$

where we used the assumption $\frac{L}{\mu} \geq \frac{15375}{2}$. Substituting (B.163) and (B.164) to (B.162), we get

$$\mathbb{E}[(L1_{x_k^1 \leq 0} + \mu 1_{x_k^1 > 0})x_k^1] \leq \frac{6}{7} L y_0 - \frac{\eta L \nu}{1536} \left(\sqrt{\frac{iB}{M}} + \sqrt{j}\right),$$

which finishes the proof.

B.8.2 Proof of Lemma B.10

$$\mathbb{E}[(L1_{x_{iB+j}^1 \leq 0} + \mu 1_{x_{iB+j}^1 > 0})x_{iB+j}^1] \leq \mu \mathbb{E}[x_{iB+j}^1]$$

$$\begin{aligned}
&= \mu y_0 + \mu \mathbb{E}[x_{iB+j}^1 - y_0] \\
&\leq \mu y_0 + \mu \mathbb{E}[|x_{iB+j}^1 - x_0|] \\
&\leq \mu y_0 + \frac{1025(iB+j)\eta L\mu}{1024} y_0 + \frac{1025\eta\mu\nu}{1024} \left(\sqrt{\frac{iB}{M}} + \sqrt{j} \right),
\end{aligned}$$

where the last inequality used Lemma B.13.

B.8.3 Proof of Lemma B.11

We consider iterates within a single epoch, and hence we omit the subscripts denoting epochs. In our construction, each machine has the same set of component functions, that is, there will be no inter-machine deviation. We therefore omit the superscript m from the local component functions. Consider the function

$$\begin{aligned}
G_2(x) &:= \frac{1}{N} \left(\sum_{i=1}^{\frac{N}{2}} g_{+1}(x) + \sum_{i=\frac{N}{2}+1}^N g_{-1}(x) \right), \text{ where} \\
g_{+1}(x) &:= \frac{Lx^2}{2} + \nu x, \text{ and } g_{-1}(x) := \frac{Lx^2}{2} - \nu x.
\end{aligned}$$

Hence, $G_2(x) = \frac{Lx^2}{2}$. We prove the lemma by coupling iterates corresponding to F_2 and G_2 . In particular, we perform local RR on F_2 and G_2 such that both start the given epoch at y_0 and all the corresponding machines use the same random permutations. Let $x_{iB+j,F}^m$ and $x_{iB+j,G}^m$ denote the iterates (for $(iB+j)$ -th iteration at machine m) for F_2 and G_2 respectively. We use mathematical induction to prove that $x_{iB+j,F}^m \geq x_{iB+j,G}^m$ for all $i = 0, \dots, \frac{N}{B} - 1$ and $j = 0, \dots, B$ and machines $m = 1, \dots, M$. After that, we will use this to prove our desired statement $\mathbb{E}[y_{\frac{N}{B},F} \mid y_0 < 0] = \mathbb{E}[\frac{1}{M} \sum_{m=1}^M x_{N,F}^m \mid y_0 < 0] \geq (1 - \frac{7\eta LN}{8B})y_0$.

Let σ^m be a random permutation of $\frac{N}{2} + 1$'s and $\frac{N}{2} - 1$'s. First we consider $i = 0$ and $0 \leq j \leq B$.

Base case. For the base case, we know that $x_{0,F}^m \geq x_{0,G}^m$, since $x_{0,F}^m = x_{0,G}^m = y_0$ for all m .

Inductive case. There can be three cases:

- Case 1: $x_{iB+j,F}^m \geq x_{iB+j,G}^m \geq 0$. Then,

$$\begin{aligned}
&x_{iB+j+1,F}^m - x_{iB+j+1,G}^m \\
&= x_{iB+j,F}^m - x_{iB+j,G}^m - \eta(\nabla f_{\sigma_{iB+j+1}^m}(x_{iB+j,F}) - \nabla g_{\sigma_{iB+j+1}^m}(x_{iB+j,G}))
\end{aligned}$$

$$\begin{aligned}
&= x_{iB+j,F}^m - x_{iB+j,G}^m - \eta \left(\mu x_{iB+j,F}^m + \nu \sigma_{iB+j+1}^m - L x_{iB+j,G}^m - \nu \sigma_{iB+j+1}^m \right) \\
&= x_{iB+j,F}^m - x_{iB+j,G}^m - \eta \left(\mu x_{iB+j,F}^m - L x_{iB+j,G}^m \right) \\
&= x_{iB+j,F}^m (1 - \eta \mu) - x_{iB+j,G}^m (1 - \eta L) \geq 0.
\end{aligned}$$

- Case 2: $0 \geq x_{iB+j,F}^m \geq x_{iB+j,G}^m$. Then,

$$\begin{aligned}
&x_{iB+j+1,F}^m - x_{iB+j+1,G}^m \\
&= x_{iB+j,F}^m - x_{iB+j,G}^m - \eta (\nabla f_{\sigma_{iB+j+1}^m}(x_{iB+j,F}) - \nabla g_{\sigma_{iB+j+1}^m}(x_{iB+j,G})) \\
&= x_{iB+j,F}^m - x_{iB+j,G}^m - \eta \left(L x_{iB+j,F}^m + \nu \sigma_{iB+j+1}^m - L x_{iB+j,G}^m - \nu \sigma_{iB+j+1}^m \right) \\
&= x_{iB+j,F}^m - x_{iB+j,G}^m - \eta \left(L x_{iB+j,F}^m - L x_{iB+j,G}^m \right) \\
&= x_{iB+j,F}^m (1 - \eta L) - x_{iB+j,G}^m (1 - \eta L) \geq 0.
\end{aligned}$$

- Case 3: $x_{iB+j,F} \geq 0 \geq x_{iB+j,G}$. Then,

$$\begin{aligned}
&x_{iB+j+1,F}^m - x_{iB+j+1,G}^m \\
&= x_{iB+j,F}^m - x_{iB+j,G}^m - \eta (\nabla f_{\sigma_{iB+j+1}^m}(x_{iB+j,F}) - \nabla g_{\sigma_{iB+j+1}^m}(x_{iB+j,G})) \\
&= x_{iB+j,F}^m - x_{iB+j,G}^m - \eta \left(\mu x_{iB+j,F}^m + \nu \sigma_{iB+j+1}^m - L x_{iB+j,G}^m - \nu \sigma_{iB+j+1}^m \right) \\
&= x_{iB+j,F}^m - x_{iB+j,G}^m - \eta \left(\mu x_{iB+j,F}^m - L x_{iB+j,G}^m \right) \\
&= x_{iB+j,F}^m (1 - \eta \mu) - x_{iB+j,G}^m (1 - \eta L).
\end{aligned}$$

Note that since $\eta \leq \frac{1}{LN}$, we get that $x_{iB+j,F}^m (1 - \eta \mu) \geq 0$ and $x_{iB+j,G}^m (1 - \eta L) \leq 0$, which proves that $x_{iB+j+1,F}^m - x_{iB+j+1,G}^m \geq 0$.

Thus, we see that $x_{iB+j+1,F}^m \geq x_{iB+j+1,G}^m$ for all the three cases, which proves by mathematical induction that $x_{iB+j,F}^m \geq x_{iB+j,G}^m$ for all $0 \leq j \leq B$ and $i = 0$. Note that this implies that, the aggregated averages $y_{1,F} := \frac{1}{M} \sum_{m=1}^M x_{B,F}^m$ and $y_{1,G} := \frac{1}{M} \sum_{m=1}^M x_{B,G}^m$ satisfy $y_{1,F} \geq y_{1,G}$. Hence, after synchronization is complete, we get that for $i = 1$ and $j = 0$, $x_{iB+j,F}^m \geq x_{iB+j,G}^m$ for all machines m . This proves the base case for $i = 1$. Now, we can repeat the Inductive cases for $1 \leq j \leq B$ and $i = 1$, and thereby prove that $y_{2,F} \geq y_{2,G}$. Continuing on this process, we get that $x_{iB+j,F}^m \geq x_{iB+j,G}^m$ for all $0 \leq j \leq B$ and $0 \leq i \leq \frac{N}{B} - 1$, and consequently, $y_{\frac{N}{B},F} \geq y_{\frac{N}{B},G}$. Further, by linearity of expectation and gradient, it is easy to check that for any machine m ,

$$\mathbb{E}[y_{\frac{N}{B},G}] = (1 - \eta L)^N y_0.$$

Using the result that $y_{\frac{N}{B},F} \geq y_{\frac{N}{B},G}$ which we proved above, we get $\mathbb{E}[y_{\frac{N}{B},F}] \geq (1 - \eta L)^N y_0$ for any initial iterate y_0 . Specifically for $y_0 < 0$, this implies $\mathbb{E}[y_{\frac{N}{B},F} \mid y_0 < 0] \geq (1 - \eta L)^N y_0$.

Further, since $\eta L \leq \frac{1}{1025N}$, we have $(1 - \eta L)^N \leq 1 - \frac{7\eta LN}{8}$. This is because $1 - \frac{7zN}{8} - (1 - z)^N$ is nonnegative on the interval $[0, 1 - (7/8)^{\frac{1}{N-1}}]$, and $1 - (7/8)^{\frac{1}{N-1}} \geq \frac{1}{1025N}$ for all $N \geq 2$. To see why, note that $(1 - \frac{1}{1025(n-1)})^{n-1} \geq \frac{7}{8}$ for all $n \geq 2$, and this gives $1 - (7/8)^{\frac{1}{n-1}} \geq \frac{1}{1025(n-1)}$, which then implies $1 - (7/8)^{\frac{1}{n-1}} \geq \frac{1}{1025n}$ for all $n \geq 2$. Therefore, for $y_0 < 0$, we have $\mathbb{E}[y_{\frac{N}{B},F} \mid y_0 < 0] \geq (1 - \frac{7\eta LN}{8})y_0$.

For the last statement of the lemma, note that by symmetry of the function G_2 , if we initialize Algorithm 5.1 at 0, then for any starting iterate of an epoch we have $\mathbb{P}(y_{0,G} \geq 0) \geq 1/2$. This combined with the fact that $y_{i,F} \geq y_{i,G}$ gives us that $\mathbb{P}(y_{0,F} \geq 0) \geq 1/2$.

B.8.4 Proof of Lemma B.13

$$\begin{aligned}
\mathbb{E}[|x_{iB+j}^1 - y_0|] &= \mathbb{E} \left[\left| \frac{\eta}{M} \sum_{m=1}^M \sum_{l=0}^{iB-1} \nu \sigma_{l+1}^m + (L1_{x_l^m < 0} + \mu 1_{x_l^m \geq 0}) x_l^m \right. \right. \\
&\quad \left. \left. + \eta \sum_{l=iB}^{iB+j-1} \nu \sigma_{l+1}^1 + (L1_{x_l^1 < 0} + \mu 1_{x_l^1 \geq 0}) x_l^1 \right| \right] \\
&\leq \eta \nu \left(\sqrt{\frac{iB}{M}} + \sqrt{j} \right) + \eta \mathbb{E} \left[\left| \sum_{l=0}^{iB+j-1} (L1_{x_l^1 < 0} + \mu 1_{x_l^1 \geq 0}) x_l^1 \right| \right] \quad (\text{By Lemma B.7}) \\
&\leq \eta \nu \left(\sqrt{\frac{iB}{M}} + \sqrt{j} \right) + \eta L \sum_{l=0}^{iB+j-1} \mathbb{E}[|x_l^1|] \\
&\leq \eta \nu \left(\sqrt{\frac{iB}{M}} + \sqrt{j} \right) + (iB + j)\eta L y_0 + \eta L \sum_{l=0}^{iB+j-1} \mathbb{E}[|x_l^1 - y_0|].
\end{aligned}$$

Now define

$$h(k) := \eta \nu \left(\sqrt{\frac{B \lfloor k/B \rfloor}{M}} + \sqrt{k - B \lfloor k/B \rfloor} \right) + k\eta L y_0 + \eta L \sum_{l=0}^{k-1} h(l).$$

In terms of i and j , note that k corresponds to $k = iB + j$. Then using induction, it can be seen that $\mathbb{E}[|x_k^1 - y_0|] \leq h(k)$. Further, since $h(k)$ is an increasing function of k , we get

$$\begin{aligned}
h(k) &= \eta \nu \left(\sqrt{\frac{B \lfloor k/B \rfloor}{M}} + \sqrt{k - B \lfloor k/B \rfloor} \right) + k\eta L y_0 + \eta L \sum_{l=0}^{k-1} h(l) \\
&\leq \eta \nu \left(\sqrt{\frac{B \lfloor k/B \rfloor}{M}} + \sqrt{k - B \lfloor k/B \rfloor} \right) + k\eta L y_0 + k\eta L h(k)
\end{aligned}$$

$$\implies h(k) \leq \frac{\eta\nu \left(\sqrt{\frac{B\lfloor k/B \rfloor}{M}} + \sqrt{k - B\lfloor k/B \rfloor} \right) + k\eta Ly_0}{1 - k\eta L}.$$

Since $k \leq N$ and $\eta \leq \frac{1}{1025LN}$, we get that

$$\mathbb{E}[|x_{iB+j}^1 - y_0|] \leq \frac{1025}{1024} \eta\nu \left(\sqrt{\frac{iB}{M}} + \sqrt{j} \right) + \frac{1025}{1024} (iB + j)\eta Ly_0,$$

as desired.

B.9 Proof of lower bound for local RR: heterogeneous case (Proposition 5.11)

Recall that Proposition 5.11 gives the bound for local RR in the heterogeneous setting, where different machines have different local objectives. In this section, we construct examples where there is no intra-machine variation (i.e., $f_1^m = f_2^m = \dots = f_N^m$ for all $m \in [M]$), but there is certain level of heterogeneity among different machines.

Similar to the other two lower bounds, we consider four step-size ranges and do case analysis for each of them. This time, we construct a single function F for these step-size regimes such that the convergence of local RR is “slow” for F . The final lower bound is the minimum among the lower bounds obtained for the four regimes. More concretely, we will construct a one-dimensional function $F(x)$ satisfying L -smoothness (5.1), μ -PL condition (5.2), and Assumption 5.3 such that⁹

- Local RR on $F(x)$ with $\eta \leq \frac{1}{8\mu NK}$ and initialization $y_0 = \frac{\tau}{\mu}$ results in

$$\mathbb{E}[F(y_{K, \frac{N}{B}})] = \Omega\left(\frac{\tau^2}{\mu}\right).$$

- Local RR on $F(x)$ with $\frac{1}{8\mu NK} \leq \eta \leq \frac{1}{8\mu B}$ and initialization $y_0 = 0$ results in

$$\mathbb{E}[F(y_{K, \frac{N}{B}})] = \Omega\left(\frac{\tau^2 B^2}{\mu N^2 K^2}\right).$$

⁹Again, the functions constructed in this theorem are μ -strongly convex, which is stronger than μ -PL required in Definition 5.5. Also, our functions satisfy Assumption 5.2 with $\nu = 0$.

- Local RR on $F(x)$ with $\frac{1}{8\mu B} \leq \eta \leq \frac{1}{\mu}$ and initialization $y_0 = 0$ results in

$$\mathbb{E}[F(y_{K, \frac{N}{B}})] = \Omega\left(\frac{\tau^2}{\mu}\right).$$

- Local RR on $F(x)$ with $\eta \geq \frac{1}{\mu}$ and initialization $y_0 = \frac{\tau}{\mu}$ results in

$$\mathbb{E}[F(y_{K, \frac{N}{B}})] = \Omega\left(\frac{\tau^2}{\mu}\right).$$

In the subsequent subsections, we prove the lower bounds for F for the four step-size intervals.

B.9.1 Lower bound for $\frac{1}{8\mu NK} \leq \eta \leq \frac{1}{8\mu B}$ and $\frac{1}{8\mu B} \leq \eta \leq \frac{1}{\mu}$

We first consider the two intervals in the middle, because they are more interesting cases. The global objective function F and its local objective functions are as follows.

$$F(x) := \frac{1}{M} \left(\sum_{i=1}^{\frac{M}{2}} f_1(x) + \sum_{i=\frac{M}{2}+1}^M f_2(x) \right), \text{ where}$$

$$f_1(x) := -\tau x, \text{ and } f_2(x) := \mu x^2 + \tau x$$

In this construction, $M/2$ machines will have the function f_1 as their N local component functions (and hence their local objective functions) and the other $M/2$ machines will have the function f_2 .

Then, B local RR updates in each machine corresponds to B updates using either f_1 or f_2 . If we start from $x_{iB}^m = y_i$, the B local updates on machine m result in

$$x_{(i+1)B}^m = \begin{cases} x_{iB}^m + \eta\tau B & \text{if machine } m \text{ has } f_1, \\ (1 - 2\eta\mu)^B x_{iB}^m - \eta\tau \sum_{j=0}^{B-1} (1 - 2\eta\mu)^j & \text{if machine } m \text{ has } f_2. \end{cases}$$

Taking the average of the M machines, we get that

$$\begin{aligned} y_{i+1} &= \frac{1}{M} \left(\frac{M}{2} (y_i + \eta\tau B) + \frac{M}{2} \left((1 - 2\eta\mu)^B y_i - \eta\tau \sum_{j=0}^{B-1} (1 - 2\eta\mu)^j \right) \right) \\ &= \frac{1}{2} \left(1 + (1 - 2\eta\mu)^B \right) y_i + \frac{\eta\tau}{2} \left(B - \sum_{j=0}^{B-1} (1 - 2\eta\mu)^j \right). \end{aligned}$$

Since there are total $\frac{NK}{B}$ such communication rounds over K epochs, at the end of the run we have

$$\begin{aligned}
y_{K, \frac{N}{B}} &= \left(\frac{1}{2} \left(1 + (1 - 2\eta\mu)^B \right) \right)^{\frac{NK}{B}} y_0 \\
&\quad + \frac{\eta\tau}{2} \left(B - \sum_{j=0}^{B-1} (1 - 2\eta\mu)^j \right) \sum_{l=0}^{\frac{NK}{B}-1} \left(\frac{1}{2} \left(1 + (1 - 2\eta\mu)^B \right) \right)^l \\
&= \frac{\eta\tau}{2} \left(B - \frac{1 - (1 - 2\eta\mu)^B}{2\eta\mu} \right) \frac{1 - \left(\frac{1}{2} \left(1 + (1 - 2\eta\mu)^B \right) \right)^{\frac{NK}{B}}}{1 - \frac{1}{2} (1 + (1 - 2\eta\mu)^B)}, \tag{B.165}
\end{aligned}$$

where we used initialization $y_0 = 0$. Having defined the function and calculated its last iterate (B.165), let us now handle the two step-size regimes separately.

We first consider $\frac{1}{8\mu NK} \leq \eta \leq \frac{1}{8\mu B}$. In this case, we exploit the fact that

$$1 - 2\eta\mu B + \eta^2\mu^2 B^2 \leq (1 - 2\eta\mu)^B \leq 1 - 2\eta\mu B + 4\eta^2\mu^2 B^2, \tag{B.166}$$

when $0 \leq \eta \leq \frac{1}{8\mu B}$. To see why, consider substituting $z := 2\eta\mu$. Then $h_1(z) := 1 - Bz + B^2 z^2 - (1 - z)^B$ has $h_1''(z) \geq 0$ on $z \in [0, 1]$, $h_1'(0) = 0$, and $h_1(0) = 0$, implying that $h_1(z) \geq 0$ on $z \in [0, 1]$. On the other hand, let $h_2(z) := 1 - Bz + \frac{B^2 z^2}{4} - (1 - z)^B$. If $B = 2$, then $h_2 \equiv 0$. If $B > 2$, then it can be checked that $h_2(z) \leq 0$ for small enough interval $[0, \frac{1}{4B}]$.

Using (B.166) on (B.165),

$$\begin{aligned}
y_{K, \frac{N}{B}} &= \frac{\eta\tau}{2} \left(B - \frac{1 - (1 - 2\eta\mu)^B}{2\eta\mu} \right) \frac{1 - \left(\frac{1}{2} \left(1 + (1 - 2\eta\mu)^B \right) \right)^{\frac{NK}{B}}}{1 - \frac{1}{2} (1 + (1 - 2\eta\mu)^B)} \\
&\geq \frac{\eta\tau}{2} \left(B - \frac{1 - (1 - 2\eta\mu B + \eta^2\mu^2 B^2)}{2\eta\mu} \right) \frac{1 - \left(\frac{1}{2} (1 + 1 - 2\eta\mu B + 4\eta^2\mu^2 B^2) \right)^{\frac{NK}{B}}}{1 - \frac{1}{2} (1 + 1 - 2\eta\mu B + \eta^2\mu^2 B^2)} \\
&= \frac{\eta\tau}{2} \left(\frac{\eta\mu B^2}{2} \right) \frac{1 - (1 - \eta\mu B + 2\eta^2\mu^2 B^2)^{\frac{NK}{B}}}{\eta\mu B - \frac{1}{2}\eta^2\mu^2 B^2} \\
&\geq \frac{\eta^2\mu\tau B^2}{4} \cdot \frac{1 - \left(1 - \eta\mu B + \frac{1}{4}\eta\mu B \right)^{\frac{NK}{B}}}{\eta\mu B} \\
&\geq \frac{\eta\tau B}{4} \left(1 - \left(1 - \frac{3\eta\mu B}{4} \right)^{\frac{NK}{B}} \right),
\end{aligned}$$

where we used $\eta\mu B \leq \frac{1}{8}$. Now, substituting $\eta \geq \frac{1}{8\mu NK}$ to above, we obtain

$$\begin{aligned} y_{K, \frac{N}{B}} &\geq \frac{\eta\tau B}{4} \left(1 - \left(1 - \frac{3\eta\mu B}{4} \right)^{\frac{NK}{B}} \right) \\ &\geq \frac{\tau B}{32\mu NK} \left(1 - \left(1 - \frac{3B}{32NK} \right)^{\frac{NK}{B}} \right) \\ &\geq \frac{(1 - e^{-3/32})\tau B}{32\mu NK}. \end{aligned}$$

Therefore, $F(y_{K, \frac{N}{B}}) = \Omega\left(\frac{\tau^2 B^2}{\mu N^2 K^2}\right)$.

Next, consider $\frac{1}{8\mu B} \leq \eta \leq \frac{1}{\mu}$. We take a close look at the term that appears in (B.165):

$$B - \sum_{j=0}^{B-1} (1 - 2\eta\mu)^j = B - \frac{1 - (1 - 2\eta\mu)^B}{2\eta\mu}.$$

For this term, we would like to find a lower bound which holds for all $\eta \in [\frac{1}{8\mu B}, \frac{1}{\mu}]$. To this end, consider substituting $z := 2\eta\mu$. Then, the function

$$h_3(z) := B - \sum_{j=0}^{B-1} (1 - z)^j \tag{B.167}$$

is increasing on $z \in [\frac{1}{4B}, 1]$, and we have

$$h_3\left(\frac{1}{4B}\right) = B - 4B \left(1 - \left(1 - \frac{1}{4B} \right)^B \right) \leq h_3(z), \text{ for all } z \in \left[\frac{1}{4B}, 1 \right].$$

Using $B \geq 2$, $h_3(\frac{1}{4B})$ can be lower-bounded as

$$h_3\left(\frac{1}{4B}\right) \geq B - 4B \left(1 - \left(1 - \frac{1}{8} \right)^2 \right) = \frac{B}{16}.$$

Next, for $z \geq 1$, the derivative of $h_3(z) := B - \sum_{j=0}^{B-1} (1 - z)^j = B - \frac{1 - (1 - z)^B}{z}$ is $h'_3(z) = \frac{1 - (1 - z)^{B-1}((B-1)z + 1)}{z^2}$. Since $B \geq 2$ is assumed to be even, it is easy to check that $h'_3(z) \geq 0$ for $z \geq 1$, which means that h_3 keeps increasing on $[1, 2]$. Therefore, we conclude that

$$\frac{B}{16} \leq h_3\left(\frac{1}{4B}\right) \leq h_3(z), \text{ for all } z \in \left[\frac{1}{4B}, 2 \right],$$

and hence

$$B - \sum_{j=0}^{B-1} (1 - 2\eta\mu)^j \geq \frac{B}{16},$$

for all $\frac{1}{8\mu B} \leq \eta \leq \frac{1}{\mu}$. Using $y_{K, \frac{N}{B}}$ from (B.165), we get

$$\begin{aligned} y_{K, \frac{N}{B}} &= \frac{\eta\tau}{2} \left(B - \sum_{j=0}^{B-1} (1 - 2\eta\mu)^j \right) \sum_{l=0}^{\frac{NK}{B}-1} \left(\frac{1}{2} \left(1 + (1 - 2\eta\mu)^B \right) \right)^l \\ &\geq \frac{\eta\tau}{2} \left(B - \sum_{j=0}^{B-1} (1 - 2\eta\mu)^j \right) \geq \frac{\eta\tau B}{32} \geq \frac{\tau}{256\mu}. \end{aligned}$$

Here, we used the fact that

$$\sum_{l=0}^{\frac{NK}{B}-1} \left(\frac{1}{2} \left(1 + (1 - 2\eta\mu)^B \right) \right)^l \geq \left(\frac{1}{2} \left(1 + (1 - 2\eta\mu)^B \right) \right)^0 = 1.$$

Hence, we obtain $F(y_{K, \frac{N}{B}}) = \Omega(\frac{\tau^2}{\mu})$, finishing the proof.

B.9.2 Lower bound for $\eta \leq \frac{1}{8\mu NK}$ and $\eta \geq \frac{1}{\mu}$

We now conclude with the “extreme” step-size regimes. We consider the same function F as in the previous subsection, but with a different initialization $y_0 = \frac{\tau}{\mu}$.

For F , recall from (B.165) that

$$\begin{aligned} y_{K, \frac{N}{B}} &= \left(\frac{1}{2} \left(1 + (1 - 2\eta\mu)^B \right) \right)^{\frac{NK}{B}} y_0 \\ &\quad + \frac{\eta\tau}{2} \left(B - \sum_{j=0}^{B-1} (1 - 2\eta\mu)^j \right) \sum_{l=0}^{\frac{NK}{B}-1} \left(\frac{1}{2} \left(1 + (1 - 2\eta\mu)^B \right) \right)^l. \end{aligned} \quad (\text{B.168})$$

This time, we want to lower-bound the second term on the RHS of (B.168) with zero and focus on the first term. To this end, we revisit our discussion on h_3 (B.167). It is easy to check that h_3 is in fact increasing on the entire $[0, \infty)$, and $h_3(0) = 0$. This shows $B - \sum_{j=0}^{B-1} (1 - 2\eta\mu)^j \geq 0$ for any $\eta \geq 0$. Next, since B is even, $\frac{1}{2} \left(1 + (1 - 2\eta\mu)^B \right) \geq 0$ for any $\eta \geq 0$. This gives

$$y_{K, \frac{N}{B}} \geq \left(\frac{1}{2} \left(1 + (1 - 2\eta\mu)^B \right) \right)^{\frac{NK}{B}} y_0 = \left(\frac{1}{2} \left(1 + (1 - 2\eta\mu)^B \right) \right)^{\frac{NK}{B}} \frac{\tau}{\mu}. \quad (\text{B.169})$$

First, consider the interval $0 \leq \eta \leq \frac{1}{8\mu NK}$. Recall from (B.166) that for this η ,

$$(1 - 2\eta\mu)^B \geq 1 - 2\eta\mu B + \eta^2\mu^2 B^2 \geq 1 - 2\eta\mu B, \quad (\text{B.170})$$

so

$$y_{K, \frac{N}{B}} \geq \left(\frac{1}{2} \left(1 + (1 - 2\eta\mu)^B \right) \right)^{\frac{NK}{B}} \frac{\tau}{\mu} \geq (1 - \eta\mu B)^{\frac{NK}{B}} \frac{\tau}{\mu} \geq \left(1 - \frac{B}{8NK} \right)^{\frac{NK}{B}} \frac{\tau}{\mu} \geq \frac{7\tau}{8\mu},$$

since $\frac{NK}{B} \geq 1$. Hence, $F(y_{K, \frac{N}{B}}) = \Omega(\frac{\tau^2}{\mu})$.

Finally, if $\eta \geq \frac{1}{\mu}$, then we have $(1 - 2\eta\mu)^B \geq 1$, so

$$y_{K, \frac{N}{B}} \geq \left(\frac{1}{2} \left(1 + (1 - 2\eta\mu)^B \right) \right)^{\frac{NK}{B}} \frac{\tau}{\mu} \geq \frac{\tau}{\mu}.$$

As a result, $F(y_{K, \frac{N}{B}}) = \Omega(\frac{\tau^2}{\mu})$.

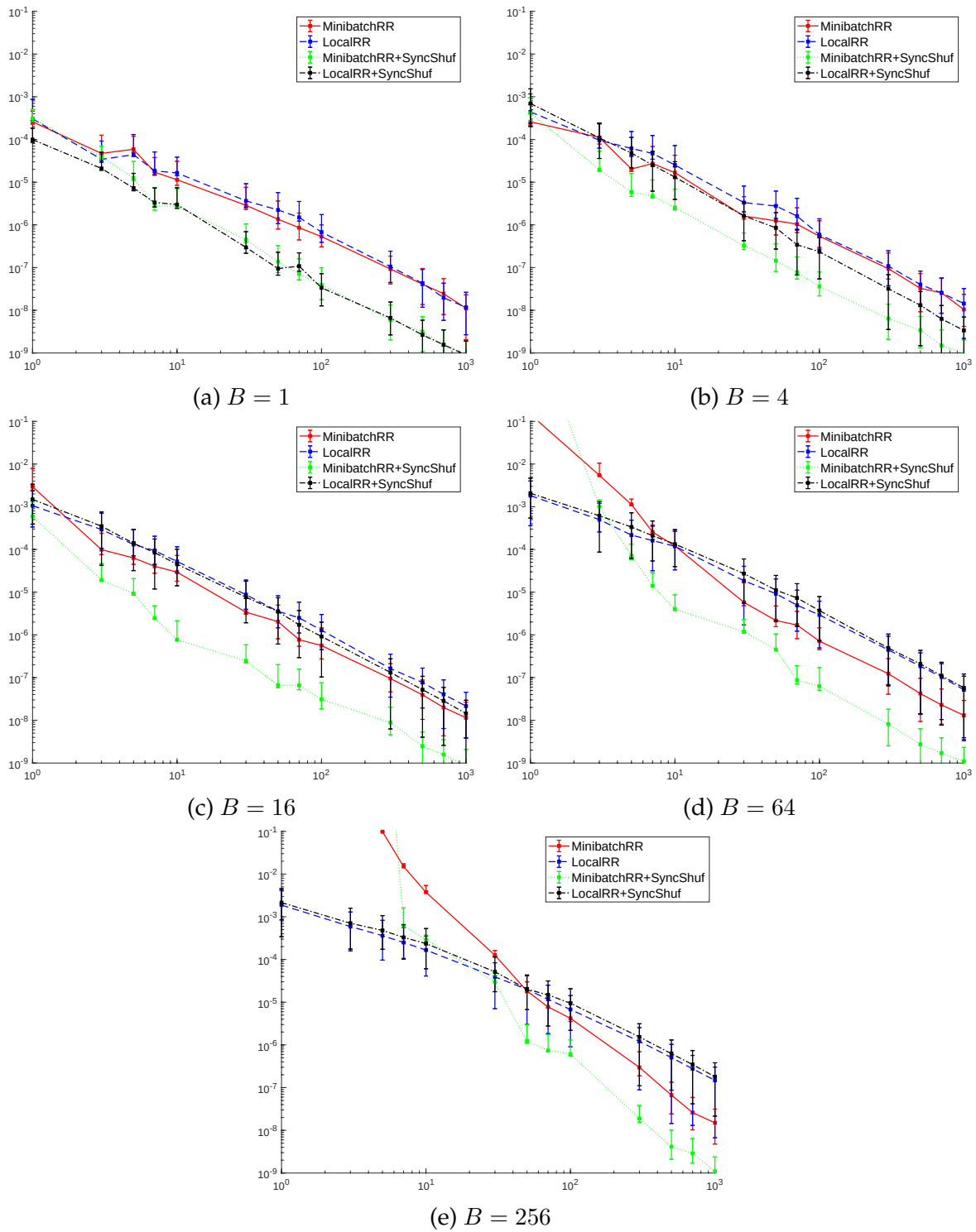


Figure B.1: Comparison between minibatch RR and local RR, with and without SYNCShuf. Best viewed in color. The algorithm versions with SYNCShuf converge faster. Also note the performance degradation as B increases, as expected by our theory.

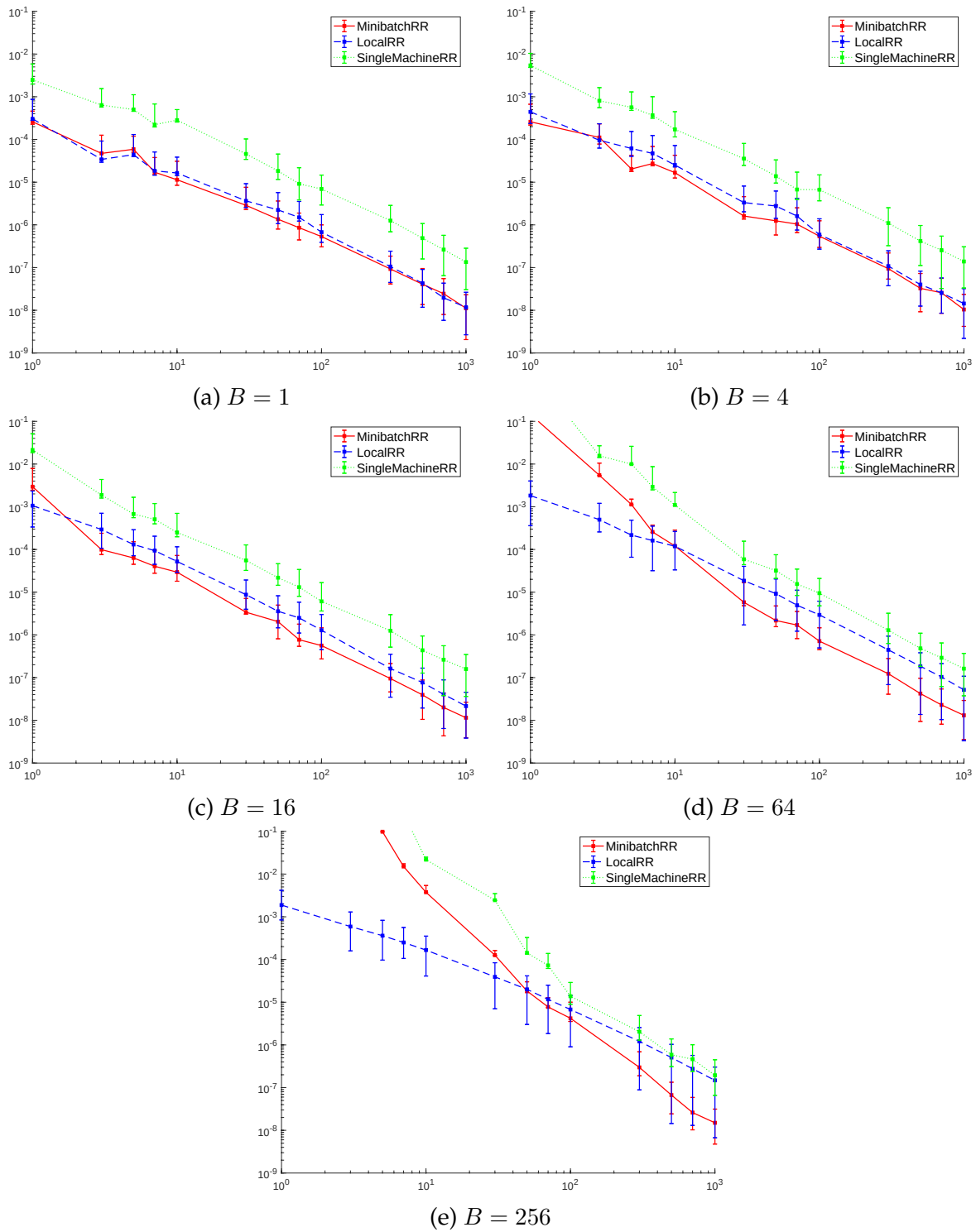


Figure B.2: Comparison of minibatch RR, local RR, and single-machine RR. Best viewed in color. The large-epoch performance of local RR becomes similar to that of single-shuffle RR as B becomes closer to N .

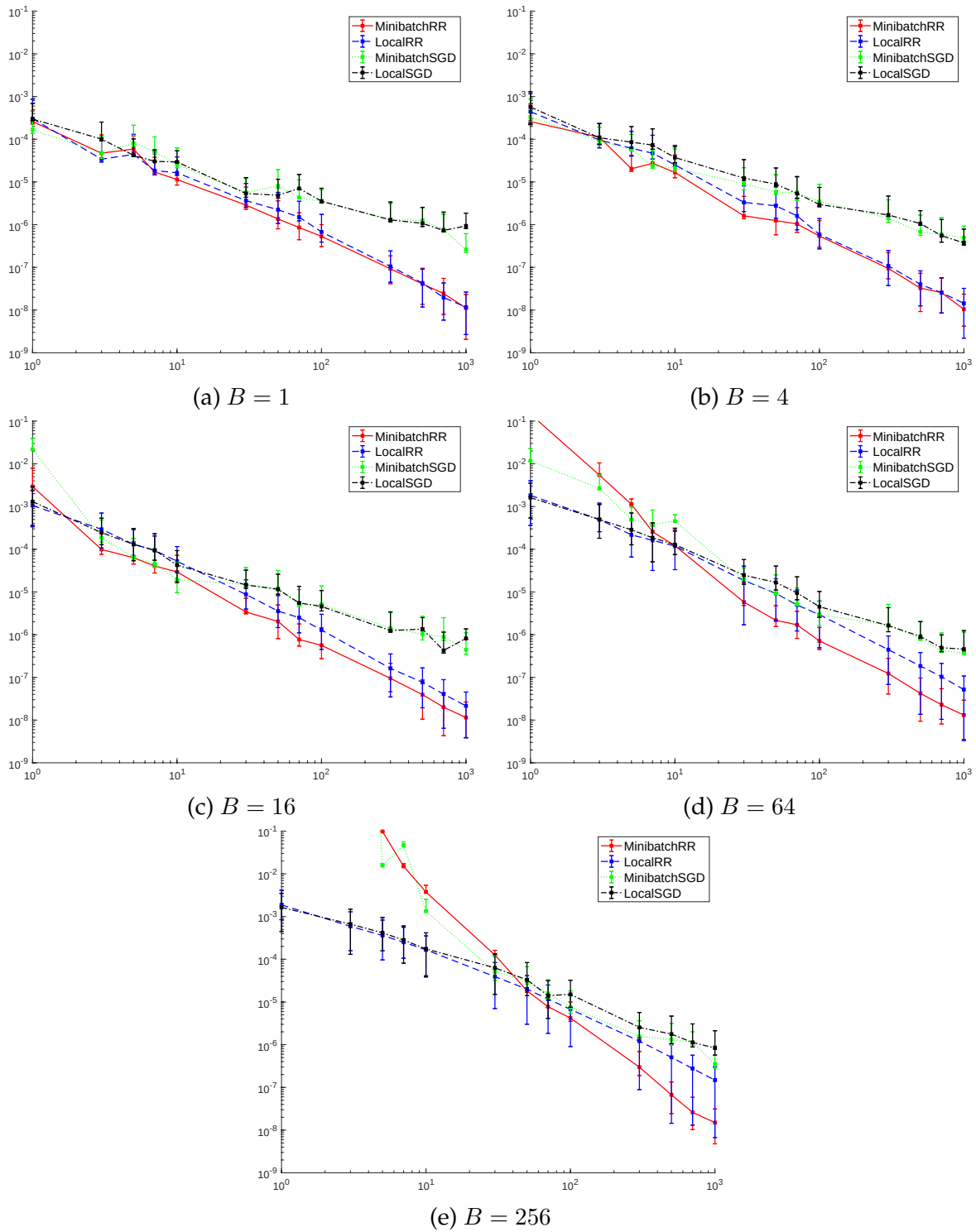


Figure B.3: Comparison of with-replacement (SGD) and without-replacement (RR) versions. Best viewed in color. Without-replacement versions converge faster than with-replacement versions, at least in our problem instance. Also note that the two versions perform similarly in the small-epoch regime, which supports our theoretical findings.

C APP: RANDOM RESHUFFLING: IS RANDOM OPTIMAL?

C.1 Discussion and Future Work

Being the first paper (to the best of our knowledge) to theoretically analyze the optimality of random permutations, we limited our scope to a specific, but common theoretical setting - strongly convex functions with constant step size. We think that future work can generalize the results of this paper to settings of non-convexity, variable step sizes, as well as techniques like variance reduction, momentum, etc.

C.1.1 Lower bounds for variable step sizes

All the existing lower bounds (to the best of our knowledge) work in the constant step size regime (Safran and Shamir (2020); Rajput et al. (2020); Safran and Shamir (2021)). Thus, generalizing the lower bounds to variable step size algorithms would be a very important direction for future research.

However, the case when step sizes are not constant can be tricky to prove lower bounds, since the step size could potentially depend on the permutation, and the current iterate. A more reasonable setting to prove lower bounds could be the case when the step sizes follow a schedule over epochs, similar to what happens in practice.

C.1.2 Faster permutations for non-quadratic objectives

The analysis of FlipFlop leverages the fact that the Hessians of quadratic functions are constant. We think that the analysis of FlipFlop might be extended to strongly convex functions or even PL functions (which are non-convex in general), under some assumptions on the Lipschitz continuity of the Hessians, similar to how Haochen and Sra (2019a) extended their analysis of quadratic functions to more general classes. A key take-away from FlipFlop is that we had to understand how permutation based SGD works specifically for quadratic functions, that is, we did a white-box analysis. In general, we feel that depending on the specific class of non-convex functions (say deep neural networks), practitioners would have to think about permutation-based SGD in a white-box fashion, to come up with better heuristics for shuffling.

In a concurrent work, (<https://openreview.net/pdf?id=7gWSJrP3opB>), it is shown that by greedily sorting stale gradients, a permutation order can be found which converges faster for some deep learning tasks. Hence, there do exist better permutations than random, even for deep learning tasks.

C.1.3 FlipFlop on Random Coordinate Descent

A shuffling scheme similar to FlipFlop has been used in random coordinate descent for faster practical convergence (see page 231 in Nocedal and Wright (2006)). This should be further investigated empirically and theoretically in future work. Even though the current analysis of FlipFlop does not directly go through for random coordinate descent, we think the analysis can be adapted to work.

C.2 Proof of Theorem 6.5

In certain places in the proof, we would need that $\alpha \leq \frac{1}{4nL}$. To see how this is satisfied, note that we have assumed that $\alpha \leq \frac{\mu}{8n(L^2 + L_H G)}$ in the theorem statement. Using the inequality $\mu \leq L$ in $\alpha \leq \frac{\mu}{4n(L^2 + L_H G)}$ gives that $\alpha \leq \frac{1}{4n(L + (L_H G/\mu))} \leq \frac{1}{4nL}$.

In this proof, we assume that the minimizer of F is at 0 to make the analysis simpler. This assumption can be satisfied by simply translating the origin to the minimizer (Safran and Shamir, 2020).

There are three main components in the proof:

1. Say that an epoch starts off at the minimizer. We prove that there exists at least one pair of permutations such that if we do two separate runs of the epoch, the first using the first permutation and the second using the second, then at the end of that epoch, the iterates corresponding to the two runs end up on opposite sides of the minimizer.
2. There exists a sequence of permutations and a point in the neighborhood of the minimizer, such that intializing at that point and using these sequence of permutations, we converge exactly to the minimizer.
3. Starting from any other point, we can couple the iterates with the iterates which were shown in the previous component, to get that these two sequences of iterates come close to each other exponentially fast.

We prove the first and second components in the Subsections C.2.1 and C.2.2; and conclude the proof in Subsection C.2.3 where we also prove the third component.

C.2.1 Permutations in one epoch

In this subsection, we prove that if $x_0, x_1, \dots, x_n$ are the iterates in an epoch such that $x_0 = 0$, then there exists a permutation of functions such that $x_n \geq 0$. By the same logic, we show

that there exists a permutation of functions such that $x_n \leq 0$. These will give us control over movement of iterates across epochs.

Order the gradients at the minimizer, $\nabla f_i(0)$, in decreasing order. WLOG assume that it is just $\nabla f_1, \nabla f_2, \dots, \nabla f_n$. We claim that this permutation leads to $x_n \geq 0$.

We will need the following intermediate result. Let y_i, y_{i-1} be such that $y_i = y_{i-1} - \alpha \nabla f_i(y_{i-1})$. Assume $\alpha \leq 1/L$ and $y_{i-1} \geq x_{i-1}$. Then,

$$\begin{aligned} y_i - x_i &= y_{i-1} - x_{i-1} - \alpha(\nabla f_i(y_{i-1}) - \nabla f_i(x_{i-1})) \\ &\geq y_{i-1} - x_{i-1} - \alpha L(y_{i-1} - x_{i-1}) \\ &= (1 - \alpha L)(y_{i-1} - x_{i-1}) \\ &\geq 0, \end{aligned} \tag{C.1}$$

that is, $y_{i-1} \geq x_{i-1} \implies y_i \geq x_i$.

Because 0 is the minimizer, we know that $\sum_{i=1}^n \nabla f_i(0) = 0$. Also, recall that $x_0 = 0$. There can be two cases:

1. $\forall i \in [1, n] : x_i < 0$. This cannot be true because if $\forall i : 1 \leq i \leq n-1 : x_i < 0$, then

$$x_n = \sum_{i=1}^n -\alpha \nabla f_i(x_{i-1}) \geq -\alpha \sum_{i=1}^n \nabla f_i(0) \geq 0,$$

where we used the fact that $\nabla f_i(x) \leq \nabla f_i(y)$ if $x \leq y$ and f_i is convex.

2. Thus, $\exists i \in [1, n] : x_i \geq 0$. Now, consider the sequence $y_i, y_{i+1}, \dots, y_n$ such that $y_i = 0$ and for $j \geq i+1$, $y_j = y_{j-1} - \alpha \nabla f_j(y_{j-1})$. Then because $\alpha \leq 1/L$ and $x_i \geq y_i = 0$, we get that $x_j \geq y_j$ for all $j \geq i$ (Using Ineq. (C.1)).

Hence, there is an $i \geq 1$ such that $x_i \geq y_i = 0$, and further $x_j \geq y_j$ for all $j \geq i$. Next, we repeat the process above for $y_i, \dots, y_n$. That is, there can be two cases:

1. $\forall j \in [i+1, n] : y_j < 0$. This cannot be true because if $\forall j : i+1 \leq j \leq n-1 : y_j < 0$, then

$$y_n = \sum_{j=i}^n -\alpha \nabla f_j(y_j) \geq -\alpha \sum_{j=i}^n \nabla f_j(0) \geq 0.$$

2. Thus, $\exists j \in [i+1, n] : y_j \geq 0$. Now, consider the sequence $z_j, z_{j+1}, \dots, z_n$ such that $z_j = 0$ and for $k \geq j+1$, $z_k = z_{k-1} - \alpha \nabla f_k(z_{k-1})$. Then because $\alpha \leq 1/L$ and $y_j \geq z_j = 0$, we get that $y_k \geq z_k$ for all $k \geq j$ (Using Ineq. (C.1)).

Hence, there exists an integer $j > i > 0$ such that $y_j \geq 0$. We have already proved that $x_j \geq y_j$. Thus, we have that $x_j \geq 0$. We can continue repeating this process (apply the same

two cases above for $z_j, z_{j+1}, \dots, z_n$, and so on), to get that $x_n \geq 0$. We define p to be this non-negative value of x_n . Note that the following lemma gives us that the gradients are bounded by G

Lemma C.1. Define $G^* := \max_i \|\nabla f_i(\mathbf{x}^*)\|$ and $D = \max \left\{ \|\mathbf{x}_0^1 - \mathbf{x}^*\|, \frac{G^*}{2L} \right\}$. If Assumptions 6.2 and 6.3 hold, and $\alpha < \frac{1}{8\kappa nL}$, then for any permutation-based algorithm (deterministic or random), we have

$$\begin{aligned} \forall i, j, k : \|\mathbf{x}_i^k - \mathbf{x}^*\| &\leq 2D, \quad \text{and} \\ \|\nabla f_j(\mathbf{x}_i^k)\| &\leq G^* + 2DL. \end{aligned}$$

Because the gradients are bounded by G , we get that $p \leq n\alpha G$.

Similarly, we can show that the reverse permutation leads to $x_n \leq 0$. We define q to be this non-positive value of x_n . Because we have assumed that the gradients are bounded by G , we get that $q \geq -n\alpha G$.

C.2.2 Exact convergence to the minimizer

In this section, we show that there exists a point such that if we initialize there and follow a specific permutation sequence, we land exactly at the minimizer.

In particular, we will show the following: There exists a point in $[4q, 4p]$ such that if we initialize there and follow a specific permutation sequence, then we land exactly at the minimizer.

We show this recursively. We will prove that there exists a point $m^K \in [4q, 4p]$ such that if the last epoch begins there, that is $x_0^K = m^K$, then we land at the minimizer at the end of the last epoch, that is, $x_n^K = 0$. Then, we will show that there exists a point $m^{K-1} \in [4q, 4p]$ such that if the $x_0^{K-1} = m^{K-1}$, then $x_0^K = x_n^{K-1} = m^K$. Repeating this for $K-2, \dots, 1$, we get that there exists a point $m^0 \in [4q, 4p]$ such that if we initialize the first epoch there, that is $x_0^1 = m^0$, then there is a permutation sequence such that ultimately $x_n^K = 0$.

We prove that any point $m^j \in [4q, 4p]$ can be reached at the end of an epoch by beginning the epoch at some point $m^{j-1} \in [4q, 4p]$, that is if $x_0^{j-1} = m^{j-1}$, then $x_n^{j-1} = m^j$.

- Case: $m^j \in [p, 4p]$. In this case, we show that $m^{j-1} \in [0, 4p]$. We have proved in the previous subsection that there exists a permutation σ such that if $x_0^{j-1} = 0$ then $x_n^{j-1} = p$.

Next, we have the following helpful lemma that we will also use later.

Lemma C.2. *Let $x_0, x_1, \dots, x_n$ be a sequence of iterates in an epoch and $y_0, y_1, \dots, y_n$ be another sequence of iterates in an epoch such that both use the same permutation of functions. If $\alpha \leq \frac{\mu}{2n(L^2 + L_H G)}$, then*

$$(1 - n\alpha L)|y_0 - x_0| \leq (1 - L\alpha)^n |y_0 - x_0| \leq |y_n - x_n| \leq \left(1 - \frac{1}{2}n\mu\alpha\right) |y_0 - x_0|.$$

If we set $x_0 = 4p, y_0 = 0$ in Lemma C.2 and we follow the permutation σ , then we get that

$$\begin{aligned} x_n - y_n &\in (x_0 - y_0) \left[1 - \alpha n L, 1 - \frac{\alpha n \mu}{2}\right] \\ \implies x_n - p &\in (4p - 0) \left[1 - \alpha n L, 1 - \frac{\alpha n \mu}{2}\right] \\ &\quad \text{(Since using } \sigma \text{ and } y_0 = 0 \text{ results in } y_n = p.) \\ \implies x_n &\geq 4p, \end{aligned}$$

where we used the fact that $\alpha \leq \frac{1}{4nL}$ is the last step.

Thus, if $x_0^{j-1} = 4p$ and we follow the permutation σ , then $x_n^{j-1} \geq 4p$.

Next, note that

$$\begin{aligned} x_1^{j-1} &= x_0^{j-1} - \alpha \nabla f_{\sigma(0)}(x_0^{j-1}). \\ x_2^{j-1} &= x_1^{j-1} - \alpha \nabla f_{\sigma(1)}(x_1^{j-1}) \\ &\vdots \\ x_n^{j-1} &= x_{n-1}^{j-1} - \alpha \nabla f_{\sigma(n-1)}(x_{n-1}^{j-1}) \end{aligned}$$

Looking above, we see that x_1^{j-1} is a continuous function of x_0^{j-1} ; x_2^{j-1} is a continuous function of x_1^{j-1} ; and so on. Thus, using the fact that composition of continuous functions is continuous, we get that x_n^{j-1} is also a continuous function of x_0^{j-1} . We have shown that if $x_0^{j-1} = 0$, then $x_n^{j-1} = p$ and if $x_0^{j-1} = 4p$, then $x_n^{j-1} \geq 4p$. Thus, using the fact that x_n^{j-1} is a continuous function of x_0^{j-1} , we get that for any point $m^j \in [p, 4p]$, there is at least one point $m^{j-1} \in [0, 4p]$, such that $x_0^{j-1} = m^{j-1}$ leads to $x_n^{j-1} = m^j$.

- Case: $m_j \in [4q, q]$. We can apply the same logic as above to show that there is at least one point $m^{j-1} \in [4q, 0]$, such that $x_0^{j-1} = m^{j-1} \implies x_n^{j-1} = m^j$.

- Case: $m_j \in [q, p]$. WLOG assume that $|q| < |p|$. Let σ_q be the permutation such that if $x_0^{j-1} = 0$ and the epoch uses this permutation, then we end up at $x_n^{j-1} = q$.

If we set $x_0 = 4p, y_0 = 0$ in Lemma C.2 and we follow the permutation σ_q , then we get that

$$\begin{aligned} x_n - y_n &\in (x_0 - y_0) \left[1 - \alpha n L, 1 - \frac{\alpha n \mu}{2} \right] \\ \implies x_n - q &\in (4p - 0) \left[1 - \alpha n L, 1 - \frac{\alpha n \mu}{2} \right] \\ &\quad \text{(Since using } \sigma_q \text{ and } y_0 = 0 \text{ results in } y_n = q.) \\ \implies x_n &\geq q + 4p(1 - \alpha n L) \geq q + 3p \geq 2p, \end{aligned}$$

where we used the fact that $\alpha \leq \frac{1}{4nL}$ is the last step.

Thus, if $x_0^{j-1} = 4p$ and we follow the permutation σ_q , then $x_n^{j-1} \geq 2p$.

Thus, using similar argument of continuity as the first case, we know that there is a point $m^{j-1} \in [0, 4p]$, such that $x_0^{j-1} = m^{j-1}$ leads to $x_n^{j-1} = m^j$ when we use the permutation σ_q .

C.2.3 Same sequence permutations get closer

In the previous subsection, we have shown that there exists a point $m^0 \in [4q, 4p]$ and a sequence of permutations $\sigma^1, \sigma^2, \dots, \sigma^K$ such that if $x_0^1 = m^0$ and epoch j uses permutation σ^j , then $x_n^K = 0$. In this subsection, we will show that if x_0^1 is initialized at any other point such that $|x_0^1| \leq D$, then using the same permutations $\sigma^1, \sigma^2, \dots, \sigma^K$ gives us that $|x_n^K| \leq (D + n\alpha G)e^{-K}$. For this we will repeatedly apply Lemma C.2 on all the K epochs. Assume that $x_0^1 = \nu^0$ with $|\nu^0| \leq D$.

Let y_i^j be the sequence of iterates such that $y_0^1 = m^0$ and uses the permutation sequence $\sigma^1, \sigma^2, \dots, \sigma^K$. Then, we know that $y_n^K = 0$. Let x_i^j be the sequence of iterates such that $x_0^1 = \nu^0$ and uses the same permutation sequence $\sigma^1, \sigma^2, \dots, \sigma^K$.

Then, using Lemma C.2 gives us that $|y_n^1 - x_n^1| \leq |\nu^0 - m^0|(1 - \frac{\mu\alpha n}{2})$. Thus, we get that $|y_0^2 - x_0^2| \leq |\nu^0 - m^0|(1 - \frac{\mu\alpha n}{2})$. Again applying Lemma C.2 gives us that $|y_n^2 - x_n^2| \leq |\nu^0 - m^0|(1 - \frac{\mu\alpha n}{2})^2$. Therefore, after applying it K times, we get

$$|y_n^K - x_n^K| \leq |\nu^0 - m^0| \left(1 - \frac{\mu\alpha n}{2} \right)^K.$$

Note that $|x_0^1| = |\nu^0| \leq D$, and $|y_0^1| = |m^0|$, with $m^0 \in [4q, 4p]$. We showed earlier in Subsection C.2.1 that $|p| \leq n\alpha G$ and $|q| \leq n\alpha G$. Therefore,

$$|y_n^K - x_n^K| \leq |D + 4n\alpha G| \left(1 - \frac{\mu\alpha n}{2}\right)^K.$$

Further, we know that $y_n^K = 0$. Thus,

$$\begin{aligned} |x_n^K| &\leq |D + 4n\alpha G| \left(1 - \frac{\mu\alpha n}{2}\right)^K \\ &\leq |D + 4n\alpha G| e^{-\frac{\mu\alpha n}{2}K}. \end{aligned}$$

Substituting the value of α completes the proof. Next we prove the lemmas used in this proof.

C.2.4 Proof of Lemma C.1

We restate the lemma below.

Lemma C.3. Define $G^* := \max_i \|\nabla f_i(\mathbf{x}^*)\|$ and $D = \max \left\{ \|\mathbf{x}_0^1 - \mathbf{x}^*\|, \frac{G^*}{2L} \right\}$. If Assumptions 6.2 and 6.3 hold, and $\alpha < \frac{1}{8\kappa nL}$, then for any permutation-based algorithm (deterministic or random), we have

$$\begin{aligned} \forall i, j, k : \|\mathbf{x}_i^k - \mathbf{x}^*\| &\leq 2D, \quad \text{and} \\ \|\nabla f_j(\mathbf{x}_i^k)\| &\leq G^* + 2DL. \end{aligned}$$

This lemma says that for any permutation-based algorithm, the domain of iterates and the norm of the gradient stays bounded during the optimization. This means that we can assume bounds on norm of iterates and gradients, which is not true in general for unconstrained SGD. This makes analyzing such algorithms much easier, and hence this lemma can be of independent interest for proving future results for permutation-based algorithms.

Remark C.4. This lemma does not hold in general for vanilla SGD where sampling is done with replacement. Consider the example with two functions $f_1(x) = x^2 - x$, and $f_2(x) = x$; and $F(x) = f_1(x) + f_2(x)$. This satisfies Assumptions 6.2 and 6.3, but one may choose f_2 consecutively for arbitrary many iterations, which will lead the iterates a proportional distance away from the minimizer. This kind of situation can never happen for permutation-based SGD because we see every function exactly once in every epoch and hence no particular function can attract the iterates too much towards its minimizer, and by the end of the epochs most of the noise gets cancelled out.

Note that the requirement $\alpha < \frac{1}{8\kappa nL}$ is stricter than the usual requirement $\alpha = O\left(\frac{1}{nL}\right)$, but we believe the lemma should also hold in that regime. For the current paper, however, this stronger requirement suffices.

Proof. To prove this lemma, we show two facts: If for some epoch k , $\|\mathbf{x}_0^k - \mathbf{x}^*\| \leq D$, then a) $\forall i : \|\mathbf{x}_i^k - \mathbf{x}^*\| \leq 2D$ and b) $\|\mathbf{x}_0^{k+1} - \mathbf{x}^*\| = \|\mathbf{x}_n^k - \mathbf{x}^*\| \leq D$. To see how a) and b) are sufficient to prove the lemma, assume that they are true. Then, since the first epoch begins inside the bounded region $\|\mathbf{x}_0^1 - \mathbf{x}^*\| \leq D$, we see using b) that every subsequent epoch begins inside the same bounded region, that is $\|\mathbf{x}_0^k - \mathbf{x}^*\| \leq D$ as well. Hence using a) we get that during these epochs, the iterates satisfy $\|\mathbf{x}_i^k - \mathbf{x}^*\| \leq 2D$, which is the first part of the lemma. Further, this bound together with the gradient Lipschitzness directly gives the upper bound $G^* + 2DL$ on the gradients. Thus, all we need to do to prove this lemma is to prove a) and b), which we do next.

We will prove a) and b) for $D = \max\{\|\mathbf{x}_0^1 - \mathbf{x}^*\|, \frac{2\kappa n \alpha G^*}{1-4\kappa n \alpha L}\}$. Once we do this, using $\alpha < \frac{1}{8\kappa nL}$ will give us the exact statement of the lemma.

Let $\|\mathbf{x}_0^k - \mathbf{x}^*\| \leq D$ for some epoch k . Then, we try to find the minimum number of iterations i needed so that $\|\mathbf{x}_i^k - \mathbf{x}^*\| \geq 2D$. Within this region, the gradient is bounded by $G^* + 2DL$. Thus, the minimum number of iterations needed are $\frac{2D-D}{\alpha(G^*+2DL)}$. However,

$$\begin{aligned}
\frac{2D-D}{\alpha(G^*+2DL)} &= \frac{1}{\alpha(\frac{G^*}{D} + 2L)} \\
&\geq \frac{1}{\alpha(G^* \frac{1-4\kappa n \alpha L}{2\kappa n \alpha G^*} + 2L)} && \text{(Using the fact that } D \geq \frac{2\kappa n \alpha G^*}{1-4\kappa n \alpha L} \text{.)} \\
&= \frac{1}{\alpha(\frac{1-4\kappa n \alpha L}{2\kappa n \alpha} + 2L)} \\
&= 2\kappa n \\
&\geq 2n.
\end{aligned}$$

Thus, the minimum number iterations needed to go outside the bound $\|\mathbf{x}_i^k - \mathbf{x}^*\| \geq 2D$ is more than the length of the epoch. This implies that within the epoch, $\|\mathbf{x}_i^k - \mathbf{x}^*\| \leq 2D$, which proves a).

We prove b) next:

$$\begin{aligned}
\|\mathbf{x}_n^k - \mathbf{x}^*\| &= \left\| \left(\mathbf{x}_0^k - \alpha \sum_{i=0}^{n-1} \nabla f_{\sigma_i^k}(\mathbf{x}_i^k) \right) - \mathbf{x}^* \right\| \\
&= \left\| \mathbf{x}_0^k - \mathbf{x}^* - \alpha \sum_{i=0}^{n-1} \nabla f_{\sigma_i^k}(\mathbf{x}_0^k) + \alpha \sum_{i=0}^{n-1} (\nabla f_{\sigma_i^k}(\mathbf{x}_0^k) - \nabla f_{\sigma_i^k}(\mathbf{x}_i^k)) \right\|
\end{aligned}$$

Note that $\sum_{i=0}^n \nabla f_{\sigma_i^k}(\mathbf{x}_0^k)$ is just the sum of all component gradients at $\mathbf{x}_0^k$, that is $\sum_{i=0}^n \nabla f_{\sigma_i^k}(\mathbf{x}_0^k) = n\nabla F(\mathbf{x}_0^k)$. Using this, we get

$$\begin{aligned}
\|\mathbf{x}_n^k - \mathbf{x}^*\| &= \left\| \mathbf{x}_0^k - \mathbf{x}^* - n\alpha \nabla F(\mathbf{x}_0^k) + \alpha \sum_{i=0}^n (\nabla f_{\sigma_i^k}(\mathbf{x}_0^k) - \nabla f_{\sigma_i^k}(\mathbf{x}_i^k)) \right\| \\
&\leq \left\| \mathbf{x}_0^k - \mathbf{x}^* - n\alpha \nabla F(\mathbf{x}_0^k) \right\| + \alpha \sum_{i=0}^n \left\| \nabla f_{\sigma_i^k}(\mathbf{x}_0^k) - \nabla f_{\sigma_i^k}(\mathbf{x}_i^k) \right\| \\
&\quad \text{(Triangle inequality.)} \\
&\leq \left\| \mathbf{x}_0^k - \mathbf{x}^* - n\alpha \nabla F(\mathbf{x}_0^k) \right\| + \alpha L \sum_{i=0}^n \left\| \mathbf{x}_0^k - \mathbf{x}_i^k \right\|, \tag{C.2}
\end{aligned}$$

where we used gradient Lipschitzness (Assumption 6.2) in the last step.

To bound the first term above, we use the standard analysis of gradient descent on smooth, strongly convex functions as follows

$$\begin{aligned}
\left\| \mathbf{x}_0^k - \mathbf{x}^* - n\alpha \nabla F(\mathbf{x}_0^k) \right\|^2 &= \left\| \mathbf{x}_0^k - \mathbf{x}^* \right\|^2 - 2n\alpha \langle \mathbf{x}_0^k - \mathbf{x}^*, \nabla F(\mathbf{x}_0^k) \rangle + n^2\alpha^2 \left\| \nabla F(\mathbf{x}_0^k) \right\|^2 \\
&\leq \left\| \mathbf{x}_0^k - \mathbf{x}^* \right\|^2 - 2n\alpha\mu \left\| \mathbf{x}_0^k - \mathbf{x}^* \right\|^2 + n^2\alpha^2 \left\| \nabla F(\mathbf{x}_0^k) \right\|^2 \quad \text{(Using Ineq. (6.3))} \\
&= (1 - n\alpha\mu) \left\| \mathbf{x}_0^k - \mathbf{x}^* \right\|^2 + n\alpha(n\alpha \left\| \nabla F(\mathbf{x}_0^k) \right\|^2 - \mu \left\| \mathbf{x}_0^k - \mathbf{x}^* \right\|^2) \\
&\leq (1 - n\alpha\mu) \left\| \mathbf{x}_0^k - \mathbf{x}^* \right\|^2 + n\alpha(n\alpha L^2 \left\| \mathbf{x}_0^k - \mathbf{x}^* \right\|^2 - \mu \left\| \mathbf{x}_0^k - \mathbf{x}^* \right\|^2) \\
&\quad \text{(Using gradient Lipschitzness)} \\
&= (1 - n\alpha\mu) \left\| \mathbf{x}_0^k - \mathbf{x}^* \right\|^2 + n\alpha(n\alpha L^2 - \mu) \left\| \mathbf{x}_0^k - \mathbf{x}^* \right\|^2 \\
&\leq (1 - n\alpha\mu) \left\| \mathbf{x}_0^k - \mathbf{x}^* \right\|^2,
\end{aligned}$$

where in the last step we used that $\alpha \leq \frac{\mu}{nL^2}$ since $\alpha \leq \frac{1}{8\kappa nL}$. Substituting this inequality in Ineq. (C.2), we get

$$\begin{aligned}
\|\mathbf{x}_n^k - \mathbf{x}^*\| &\leq \sqrt{1 - n\alpha\mu} \left\| \mathbf{x}_0^k - \mathbf{x}^* \right\| + \alpha L \sum_{i=0}^n \left\| \mathbf{x}_0^k - \mathbf{x}_i^k \right\| \\
&\leq \left(1 - \frac{1}{2}n\alpha\mu \right) \left\| \mathbf{x}_0^k - \mathbf{x}^* \right\| + \alpha L \sum_{i=0}^n \left\| \mathbf{x}_0^k - \mathbf{x}_i^k \right\|.
\end{aligned}$$

We have already proven a) that says that the iterates $\mathbf{x}_i^k$ satisfy $\|\mathbf{x}_i^k - \mathbf{x}^*\| \leq 2D$. Using gradient Lipschitzness, this implies that the gradient norms stay bounded by $G^* + 2DL$. Hence, $\left\| \mathbf{x}_0^k - \mathbf{x}_i^k \right\| \leq \alpha i(G^* + 2DL)$. Using this,

$$\|\mathbf{x}_n^k - \mathbf{x}^*\| \leq \left(1 - \frac{1}{2}n\alpha\mu \right) \left\| \mathbf{x}_0^k - \mathbf{x}^* \right\| + \alpha L \sum_{i=0}^n \alpha i(G^* + 2DL)$$

$$\begin{aligned}
&\leq \left(1 - \frac{1}{2}n\alpha\mu\right) D + \alpha L \sum_{i=0}^n \alpha i (G^* + 2DL) \\
&\leq \left(1 - \frac{1}{2}n\alpha\mu\right) D + n^2\alpha^2L(G^* + 2DL) \\
&\leq D,
\end{aligned}$$

where we used the fact that $D \geq \frac{2\kappa n\alpha G^*}{1-4\kappa n\alpha L}$ in the last step. $\square$

C.2.5 Proof of Lemma C.2

Without loss of generality, let $\sigma = (1, 2, 3, \dots, n)$. This is only done for ease of notation. The analysis goes through for any other permutation σ too.

First we show the lower bound. WLOG assume $y_0 > x_0$. Because $\alpha < 1/L$, we have that $\forall i, y_i > x_i$ by induction (see the equations below). Then,

$$\begin{aligned}
y_i - x_i &= y_{i-1} - x_{i-1} - \alpha(\nabla f_i(y_{i-1}) - \nabla f_i(x_{i-1})) \\
&\geq y_{i-1} - x_{i-1} - \alpha L(y_{i-1} - x_{i-1}) \\
&= (1 - \alpha L)(y_{i-1} - x_{i-1}) \\
&\quad \vdots \\
&= (1 - \alpha L)^i(y_0 - x_0) \\
&\geq (1 - i\alpha L)(y_0 - x_0).
\end{aligned} \tag{C.3}$$

Next we show the upper bound

$$\begin{aligned}
y_n - x_n &= y_0 - x_0 - \alpha \sum_{i=1}^n (\nabla f_i(y_{i-1}) - \nabla f_i(x_{i-1})) \\
&= y_0 - x_0 - \alpha \sum_{i=1}^n (\nabla f_i(y_0) - \nabla f_i(x_0)) \\
&\quad + \alpha \sum_{i=1}^n (\nabla f_i(y_0) - \nabla f_i(x_0) - \nabla f_i(y_{i-1}) + \nabla f_i(x_{i-1})) \\
&= y_0 - x_0 - n\alpha(\nabla F(y_0) - \nabla F(x_0)) \\
&\quad + \alpha \sum_{i=1}^n (\nabla f_i(y_0) - \nabla f_i(x_0) - \nabla f_i(y_{i-1}) + \nabla f_i(x_{i-1})) \\
&\leq (1 - n\alpha\mu)(y_0 - x_0) + \alpha \sum_{i=1}^n (\nabla f_i(y_0) - \nabla f_i(x_0) - \nabla f_i(y_{i-1}) + \nabla f_i(x_{i-1})).
\end{aligned}$$

(Using strong convexity)

We use the fact that the function is twice differentiable:

$$\begin{aligned}
y_n - x_n &= (1 - n\alpha\mu)(y_0 - x_0) + \alpha \sum_{i=1}^n \left(\int_{x_0}^{y_0} \nabla^2 f_i(t) dt - \int_{x_{i-1}}^{y_{i-1}} \nabla^2 f_i(t) dt \right) \\
&= (1 - n\alpha\mu)(y_0 - x_0) \\
&\quad + \alpha \sum_{i=1}^n \left(\int_{x_0}^{y_0} \nabla^2 f_i(t) dt - \int_{x_{i-1}}^{x_{i-1}+(y_0-x_0)} \nabla^2 f_i(t) dt - \int_{x_{i-1}+(y_0-x_0)}^{y_{i-1}} \nabla^2 f_i(t) dt \right) \\
&= (1 - n\alpha\mu)(y_0 - x_0) \\
&\quad + \alpha \sum_{i=1}^n \left(\int_{x_0}^{y_0} (\nabla^2 f_i(t) - \nabla^2 f_i(x_{i-1} - x_0 + t)) dt - \int_{x_{i-1}+(y_0-x_0)}^{y_{i-1}} \nabla^2 f_i(t) dt \right).
\end{aligned}$$

In the above, we used the convention that $\int_a^b f(x)dx$ is the same as $-\int_b^a f(x)dx$ if $a > b$.

Now, we can use the Hessian Lipschitzness to bound the term as follows

$$\begin{aligned}
y_n - x_n &\leq (1 - n\alpha\mu)(y_0 - x_0) \\
&\quad + \alpha \sum_{i=1}^n \left(\int_{x_0}^{y_0} L_H |x_{i-1} - x_0| dt - \int_{x_{i-1}+(y_0-x_0)}^{y_{i-1}} \nabla^2 f_i(t) dt \right) \\
&\leq (1 - n\alpha\mu)(y_0 - x_0) + \alpha \sum_{i=1}^n \left(\int_{x_0}^{y_0} L_H G \alpha n dt - \int_{x_{i-1}+(y_0-x_0)}^{y_{i-1}} \nabla^2 f_i(t) dt \right) \\
&= (1 - n\alpha\mu)(y_0 - x_0) + L_H G \alpha^2 n^2 (x_0 - y_0) - \alpha \sum_{i=1}^n \int_{x_{i-1}+(y_0-x_0)}^{y_{i-1}} \nabla^2 f_i(t) dt \\
&\leq (1 - n\alpha\mu)(y_0 - x_0) + L_H G \alpha^2 n^2 (x_0 - y_0) + \alpha \sum_{i=1}^n L((y_0 - x_0) - (y_i - x_i)) \\
&\leq (1 - n\alpha\mu)(y_0 - x_0) + L_H G \alpha^2 n^2 (x_0 - y_0) + \alpha \sum_{i=1}^n L(i\alpha L)(y_0 - x_0) \\
&\hspace{15em} \text{(Using Ineq. (C.3).)} \\
&\leq (1 - n\alpha\mu)(y_0 - x_0) + L_H G \alpha^2 n^2 (x_0 - y_0) + \alpha^2 n^2 L^2 (y_0 - x_0).
\end{aligned}$$

Thus, if we have $\alpha \leq \frac{\mu}{2n(L^2 + L_H G)}$, then

$$y_n - x_n \leq \left(1 - \frac{\mu n \alpha}{2}\right) (y_0 - x_0).$$

C.3 Proof of Theorem 6.6

To prove this theorem, we consider three step-size ranges and do a case by case analysis for each of them. We construct functions for each range such that the convergence of any permutation-based algorithm is “slow” for the functions on their corresponding step-size

regime. The final lower bound is the minimum among the lower bounds obtained for the three regimes.

In this proof, we will use different notation from the rest of the paper because we work with scalars in the proof, and hence the superscript will denote the scalar power, and not the epoch number. We will use $\mathbf{x}_{k,i}$ to denote the i -th iterate of the k -th epoch.

We will construct three functions $F_1(\mathbf{x})$, $F_2(\mathbf{y})$, and $F_3(z)$, each the means of n component functions, such that

- Any permutation-based algorithm on $F_1(\mathbf{x})$ with $\alpha \in [\frac{1}{2(n-1)KL}, \frac{3}{nL}]$ and initialization $\mathbf{x}_{1,0} = \mathbf{0}$ results in

$$\|\mathbf{x}_{K,n}\|^2 = \Omega\left(\frac{1}{n^3 K^2}\right).$$

F_1 will be an n -dimensional function, that is $\mathbf{x} \in \mathbb{R}^n$. This function will have minimizer at $\mathbf{0} \in \mathbb{R}^n$. NOTE: This is the ‘key’ step-size range, and the proof sketch explained in the main paper corresponds to this function’s construction.

- Any permutation-based algorithm on $F_2(\mathbf{y})$ with $\alpha \in [\frac{3}{nL}, \frac{1}{L}]$ and initialization $\mathbf{y}_{1,0} = \mathbf{0}$ results in

$$\|\mathbf{y}_{K,n}\|^2 = \Omega\left(\frac{1}{n^2}\right)$$

F_2 will be an n -dimensional function, that is $\mathbf{y} \in \mathbb{R}^n$. This function will have minimizer at $\mathbf{0} \in \mathbb{R}^n$. The construction for this is also inspired by the construction for F_1 , but this is constructed significantly differently due to the different step-size range.

- Any permutation-based algorithm on $F_3(z)$ with $\alpha \notin [\frac{1}{2(n-1)KL}, \frac{1}{L}]$ and initialization $z_{1,0} = 1$ results in

$$|z_{K,n}|^2 = \Omega(1)$$

F_3 will be an 1-dimensional function, that is $z \in \mathbb{R}$. This function will have minimizer at 0.

Then, the $2n+1$ -dimensional function $F([\mathbf{x}^\top, \mathbf{y}^\top, z]^\top) = F_1(\mathbf{x}) + F_2(\mathbf{y}) + F_3(z)$ will show bad convergence in any step-size regime. This function will have minimizer at $\mathbf{0} \in \mathbb{R}^{2n+1}$. Furthermore,

$$\frac{n-1}{n}L\mathbf{I} \preceq \nabla^2 F_1, \nabla^2 F_2, \nabla^2 F_3, \nabla^2 F \preceq 2L\mathbf{I},$$

that is, F_1, F_2, F_3 , and F are all $\frac{n-1}{n}L$ -strongly convex and $2L$ -smooth.

In the subsequent subsections, we prove the lower bounds for F_1, F_2 , and F_3 separately.

NOTE: In Appendix C.3.4, we have discussed how the lower bound is actually invariant to initialization.

C.3.1 Lower bound for $F_1, \alpha \in [\frac{1}{2(n-1)KL}, \frac{3}{nL}]$

We will work in n -dimensions (n is even) and represent a vector in the space as $\mathbf{z} = [x_1, y_1, \dots, x_{n/2}, y_{n/2}]$. These x_i and y_i are not related to the vectors used by F_2 and F_3 later, we only use x_i and y_i to make the proof for this part easier to understand.

We start off by defining the n component functions: For $i \in [n/2]$, define

$$f_i(\mathbf{z}) := \frac{L}{2}x_i^2 - x_i + y_i + \sum_{j \neq i} \left(\frac{L}{2}x_j^2 + \frac{L}{2}y_j^2 \right), \text{ and}$$

$$g_i(\mathbf{z}) := \frac{L}{2}y_i^2 - y_i + x_i + \sum_{j \neq i} \left(\frac{L}{2}x_j^2 + \frac{L}{2}y_j^2 \right).$$

Thus, $F(\mathbf{z}) := \frac{1}{n} \left(\sum_{i=1}^{n/2} f_i + \sum_{i=1}^{n/2} g_i \right) = \left(\frac{n-1}{n} \right) \frac{L}{2} \|\mathbf{z}\|^2$. This function has minimizer at $\mathbf{z}^* = \mathbf{0}$.

Let $\mathbf{z}_{k,j}$ denote $\mathbf{z}$ at the j -th iteration in k -th epoch. We initialize at $\mathbf{z}_{1,0} = \mathbf{0}$. For the majority of this proof, we will work inside a given epoch, so we will skip the subscript denoting the epoch. We denote the j -th iterate within the epoch as $\mathbf{z}_j$ and the coordinates as $\mathbf{z}_j = [x_{j,1}, y_{j,1}, \dots, x_{j,n/2}, y_{j,n/2}]$

In the current epoch, let σ be the permutation of $\{f_1, \dots, f_{n/2}, g_1, \dots, g_{n/2}\}$ used.

For any $i \in [n/2]$, let p and q be indices such that $\sigma_p = f_i$ and $\sigma_q = g_i$. Let us consider the case that $p < q$ (the case when $p > q$ will be analogous). Then, it can be seen that

$$\begin{aligned} x_{n,i} &= (1 - \alpha L)^{n-1} x_{0,i} + (1 - \alpha L)^{n-p-1} \alpha - (1 - \alpha L)^{n-q} \alpha, \text{ and} \\ y_{n,i} &= (1 - \alpha L)^{n-1} y_{0,i} - (1 - \alpha L)^{n-p} \alpha + (1 - \alpha L)^{n-q} \alpha. \end{aligned} \tag{C.4}$$

Hence,

$$\begin{aligned} x_{n,i} + y_{n,i} &= (1 - \alpha L)^{n-1} (x_{0,i} + y_{0,i}) + 2(1 - \alpha L)^{n-p-1} \alpha^2 L \\ &\geq (1 - \alpha L)^{n-1} (x_{0,i} + y_{0,i}) + 2(1 - \alpha L)^{n-1} \alpha^2 L. \end{aligned}$$

In the other case, when $p > q$, we will get the same inequality. Let $x_{K,n,i}$ and $y_{K,n,i}$ denote the value of $x_{n,i}$ and $y_{n,i}$ at the K -th epoch. Then, recalling that $\mathbf{z}$ was initialized to $\mathbf{0}$, we

use the inequality above to get

$$\begin{aligned}
 x_{K,n,i} + y_{K,n,i} &\geq (1 - \alpha L)^{(n-1)K} \cdot 0 + 2(1 - \alpha L)^{n-1} \alpha^2 L \frac{1 - (1 - \alpha L)^{(n-1)K}}{1 - (1 - \alpha L)^{n-1}} \\
 &= 2(1 - \alpha L)^{n-1} \alpha^2 L \frac{1 - (1 - \alpha L)^{(n-1)K}}{1 - (1 - \alpha L)^{n-1}}
 \end{aligned} \tag{C.5}$$

Since this inequality is valid for all i , we get that

$$\begin{aligned}
 \|\mathbf{z}_{K,n}\|^2 &= \sum_{i=1}^{n/2} (x_{K,n,i}^2 + y_{K,n,i}^2) \\
 &\geq \sum_{i=1}^{n/2} \frac{1}{2} (x_{K,n,i} + y_{K,n,i})^2 \\
 &\geq \sum_{i=1}^{n/2} \frac{1}{2} \left(2(1 - \alpha L)^{n-1} \alpha^2 L \frac{1 - (1 - \alpha L)^{(n-1)K}}{1 - (1 - \alpha L)^{n-1}} \right)^2 \\
 &= n \left((1 - \alpha L)^{n-1} \alpha^2 L \frac{1 - (1 - \alpha L)^{(n-1)K}}{1 - (1 - \alpha L)^{n-1}} \right)^2.
 \end{aligned} \tag{C.6}$$

Note that if $\alpha \leq \frac{3}{nL}$ and $n \geq 4$, then $(1 - \alpha L)^{n-1} \geq \frac{1}{8}$. Using this in (C.6), we get

$$\begin{aligned}
 \|\mathbf{z}_{K,n}\|^2 &\geq n \left((1 - \alpha L)^{n-1} \alpha^2 L \frac{1 - (1 - \alpha L)^{(n-1)K}}{1 - (1 - \alpha L)^{n-1}} \right)^2 \\
 &\geq \frac{n}{128} \alpha^4 L^2 \left(\frac{1 - (1 - \alpha L)^{(n-1)K}}{1 - (1 - \alpha L)^{n-1}} \right)^2 \\
 &= \frac{n}{128 L^2} \left(\frac{(\alpha L)^2}{1 - (1 - \alpha L)^{n-1}} \right)^2 \left(1 - (1 - \alpha L)^{(n-1)K} \right)^2
 \end{aligned}$$

We consider two cases:

1. **Case A**, $\alpha L \leq \frac{1}{2(n+2)}$: It can be verified that $\frac{(\alpha L)^2}{1 - (1 - \alpha L)^{n-1}}$ is an increasing function of α when $\alpha L \leq \frac{1}{2(n+2)}$. Noting that we are working in the range when $\alpha \geq \frac{1}{2(n-1)KL}$, then

$$\begin{aligned}
 \|\mathbf{z}_{K,n}\|^2 &\geq \frac{n}{128 L^2} \left(\frac{\left(\frac{1}{2(n-1)K} \right)^2}{1 - \left(1 - \frac{1}{2(n-1)K} \right)^{n-1}} \right)^2 \left(1 - (1 - \alpha L)^{(n-1)K} \right)^2 \\
 &\geq \frac{n}{128 L^2} \left(\frac{\left(\frac{1}{2(n-1)K} \right)^2}{\frac{1}{2(n-1)K} (n-1)} \right)^2 \left(1 - (1 - \alpha L)^{(n-1)K} \right)^2
 \end{aligned}$$

$$\begin{aligned}
&= \frac{n}{512(n-1)^4 K^2 L^2} \left(1 - (1 - \alpha L)^{(n-1)K}\right)^2 \\
&\geq \frac{n}{512(n-1)^4 K^2 L^2} \left(1 - e^{-\alpha L(n-1)K}\right)^2 \\
&\geq \frac{n}{512(n-1)^4 K^2 L^2} \left(1 - e^{-1/2}\right)^2 \\
&= \Omega\left(\frac{1}{n^3 K^2 L^2}\right).
\end{aligned}$$

2. Case B, $\alpha L > \frac{1}{2(n+2)}$: In this case,

$$\begin{aligned}
\|\mathbf{z}_{K,n}\|^2 &\geq \frac{n}{128L^2} \left(\frac{\left(\frac{1}{2(n+2)}\right)^2}{1}\right)^2 \left(1 - (1 - \alpha L)^{(n-1)K}\right)^2 \\
&\geq \frac{n}{128L^2} \left(\frac{\left(\frac{1}{2(n+2)}\right)^2}{1}\right)^2 \left(1 - e^{-\alpha L(n-1)K}\right)^2 \\
&\geq \frac{n}{128L^2} \left(\frac{\left(\frac{1}{2(n+2)}\right)^2}{1}\right)^2 \left(1 - e^{-\frac{3(n-1)K}{2(n+2)}}\right)^2 \\
&= \Omega\left(\frac{1}{n^3 L^2}\right).
\end{aligned}$$

C.3.2 Lower bound for F_2 , $\alpha \in [\frac{3}{nL}, \frac{1}{L}]$

We will work in n -dimensions and represent a vector in the space as $\mathbf{y} = [y_1, \dots, y_n]$.

We start off by defining the n component functions: For $i \in [n]$, define

$$f_i(\mathbf{y}) := -y_i + \sum_{j \neq i} \left(\frac{L y_j^2}{2} + \frac{y_j}{n-1} \right)$$

Thus, $F(\mathbf{y}) := \frac{1}{n} \sum_{i=1}^n f_i = \left(\frac{n-1}{n}\right) \frac{L}{2} \|\mathbf{y}\|^2$. This function has minimizer at $\mathbf{y}^* = \mathbf{0}$.

Let $\mathbf{y}_{k,j}$ denote $\mathbf{y}$ at the j -th iteration in k -th epoch. We initialize at $\mathbf{y}_{1,0} = \mathbf{0}$. For the majority of this proof, we will work inside a given epoch. We denote the j -th iterate within the epoch as $\mathbf{y}_j$ and the coordinates as $\mathbf{y}_j = [y_{j,1}, \dots, y_{j,n}]$

In the current epoch, let σ be the permutation of $\{f_1, \dots, f_n\}$ used. Let i be the index such that $\sigma_n = i$, that is, i is the last element of the permutation σ . Then at the end of the epoch,

$$y_{n,i} = (1 - \alpha L)^{n-1} y_{0,i} + \alpha - \frac{\alpha}{n-1} \sum_{j=0}^{n-2} (1 - \alpha L)^j$$

$$= (1 - \alpha L)^{n-1} y_{0,i} + \alpha - \frac{(1 - (1 - \alpha L)^{n-1})}{(n-1)L}. \quad (\text{C.7})$$

For some $j \in [n]$, let s be the integer such that $\sigma_s = j$, that is j is the s -th element in the permutation σ . Then for any j and any epoch,

$$y_{n,j} = (1 - \alpha L)^{n-1} y_{0,j} + \alpha(1 - \alpha L)^{n-s} - \frac{\alpha}{n-1} \sum_{j=0}^{n-2} (1 - \alpha L)^j.$$

Then,

$$\begin{aligned} y_{n,j} &\geq (1 - \alpha L)^{n-1} y_{0,j} - \frac{\alpha}{n-1} \sum_{j=0}^{n-2} (1 - \alpha L)^j \\ &= (1 - \alpha L)^{n-1} y_{0,j} - \frac{(1 - (1 - \alpha L)^{n-1})}{(n-1)L}. \end{aligned}$$

Note that the above is independent of σ , and hence applicable for all epochs. Applying it recursively and noting that we initialized $\mathbf{y}_{1,0} = \mathbf{0}$, we get

$$\begin{aligned} y_{n,j} &\geq -\frac{(1 - (1 - \alpha L)^{n-1})}{(n-1)L} \sum_{k=1}^K (1 - \alpha L)^n \\ &\geq -\frac{1}{(n-1)L}. \end{aligned}$$

Note that $y_{0,i}$ is just $y_{n,i}$ from the previous epoch. Hence we can substitute the inequality above for $y_{0,i}$ in (C.7). Thus,

$$\begin{aligned} y_{n,i} &\geq -\frac{(1 - \alpha L)^{n-1}}{(n-1)L} + \alpha - \frac{(1 - (1 - \alpha L)^{n-1})}{(n-1)L} \\ &= \alpha - \frac{1}{(n-1)L} \\ &\geq \frac{3}{nL} - \frac{1}{(n-1)L} \\ &= \Omega\left(\frac{1}{nL}\right). \end{aligned}$$

This gives us that $\|\mathbf{y}_{k,n}\|^2 = \Omega\left(\frac{1}{n^2 L^2}\right)$ for any k .

C.3.3 Lower bound for F_3 , $\alpha \notin [\frac{1}{2(n-1)KL}, \frac{1}{L}]$

Consider the function $F(z) = \frac{1}{n} \sum_{i=1}^n f_i(z)$, where $f_i(z) = Lz^2$ for all i . Note that the gradient of any function at z is just $-2Lz$. Hence, regardless of the permutation, if we start

the shuffling based gradient descent method at $z_{1,0} = 1$, we get that

$$z_{K,n} = (1 - 2\alpha L)^{nK} z_{1,0} = (1 - 2\alpha L)^{nK}.$$

In the case when $\alpha \leq \frac{1}{2(n-1)KL}$, we see that

$$\begin{aligned} z_{K,n} &\geq \left(1 - 2\frac{1}{2(n-1)KL}L\right)^{nK} \\ &\geq \left(1 - \frac{1}{(n-1)K}\right)^{nK} \\ &= \Omega(1), \end{aligned}$$

for $n, K \geq 2$. Finally, in the case when $\alpha \geq \frac{1}{L}$, we see that

$$\begin{aligned} |z_{K,n}| &= |1 - 2\alpha L|^{nK} \\ &\geq \left|1 - 2\frac{1}{L}L\right|^{nK} \\ &\geq 1^{nK} \\ &= \Omega(1). \end{aligned}$$

C.3.4 Discussion about initialization

The lower bound partitions the step size in three ranges -

- In the step size ranges $\alpha \in [\frac{1}{2(n-1)KL}, \frac{3}{nL}]$ and $\alpha \in [\frac{3}{nL}, \frac{1}{L}]$, the initializations are done at the minimizer and it is shown that any permutation-based algorithm will still move away from the minimizer. The choice of initializing at the minimizer was solely for the convenience of analysis and calculations, and the proof should work for any other initialization as well.

Furthermore, the effect of initializing at any arbitrary (non-zero) point will decay exponentially fast with epochs anyway. To see how, note that every epoch can be treated as n steps of full gradient descent and some noise, and hence the full gradient descent part will essentially keep decreasing the effect of initialization exponentially, and what we would be left with is the noise in each epoch. Thus, it was more convenient for us to just assume initialization at the minimizer and only focus on the noise in each epoch.

- The step size range $\alpha \notin [\frac{1}{2(n-1)KL}, \frac{1}{L}]$ can be divided into two parts, $\alpha \in [0, \frac{1}{2(n-1)KL}]$ and $\alpha \in [\frac{1}{L}, \infty)$.

For the range $\alpha \in [0, \frac{1}{2(n-1)KL}]$, we essentially show that the step size is too small to make any meaningful progress towards the minimizer. Hence, instead of initializing at 1, initializing at any other arbitrary (non-zero) point will also give the same slow convergence rate.

For the range $\alpha \in [\frac{1}{L}, \infty)$, we show that the optimization algorithm will in fact diverge since the step size is too large. Hence, even here, any other arbitrary (non-zero) point of initialize will also give divergence.

C.4 Proof of Theorem 6.7

Define $f_1(x) := \frac{L}{2}x^2 - x$ and $f_2(x, y) := -\frac{L}{4}x^2 + x$. Thus, $F(x, y) = \frac{L}{8}x^2$. This function has minimizer at $x^* = 0$. For this proof, we will use the convention that $x_{i,j}$ is the iterate after the i -th iteration of the j -th epoch. Further, the number in the superscript will be the scalar power. For example $x_{i,j}^2 = x_{i,j} \cdot x_{i,j}$.

Initialize at $x_{0,0} = \frac{1}{L}$. Then at epoch k , there are two possible permutations: $\sigma = (1, 2)$ and $\sigma = (2, 1)$. In the first case, $\sigma = (1, 2)$, we get that after the first iteration of the epoch,

$$\begin{aligned} x_{1,k} &= x_{0,k} - \alpha \nabla f_1(x_{0,k}, y_{0,k}) \\ &= (1 - \alpha L)x_{0,k} + \alpha, \end{aligned}$$

Continuing on, in the second iteration, we get

$$\begin{aligned} x_{2,k} &= x_{1,k} - \alpha \nabla f_2(x_{1,k}, y_{1,k}) \\ &= \left(1 + \frac{1}{2}\alpha L\right) x_{1,k} - \alpha \\ &= \left(1 + \frac{1}{2}\alpha L\right) ((1 - \alpha L)x_{0,k} + \alpha) - \alpha \\ &= \left(1 + \frac{1}{2}\alpha L\right) (1 - \alpha L)x_{0,k} + \frac{1}{2}\alpha^2 L. \end{aligned}$$

Note that $x_{0,k+1} = x_{2,k}$. Thus, $x_{0,k+1} = \left(1 + \frac{1}{2}\alpha L\right) (1 - \alpha L)x_{0,k} + \frac{1}{2}\alpha^2 L$.

Similarly, for the other possible permutation, $\sigma = (2, 1)$, we get $x_{0,k+1} = \left(1 + \frac{1}{2}\alpha L\right) (1 - \alpha L)x_{0,k} + \alpha^2 L$. Thus, regardless of what permutation we use, we get that

$$x_{0,k+1} \geq \left(1 + \frac{1}{2}\alpha L\right) (1 - \alpha L)x_{0,k} + \frac{1}{2}\alpha^2 L \geq (1 - \alpha L)x_{0,k} + \frac{1}{2}\alpha^2 L.$$

Hence, recalling that we initialized at $x_{0,0} = \frac{1}{L}$, we get

$$\begin{aligned}
 x_{n,K} &= x_{0,K+1} \\
 &\geq (1 - \alpha L)^K \frac{1}{L} + \frac{1}{2} \alpha^2 L \sum_{i=0}^{K-1} (1 - \alpha L)^i \\
 &= \frac{1}{L} (1 - \alpha L)^K + \frac{1}{2} \alpha^2 L \frac{1 - (1 - \alpha L)^K}{1 - (1 - \alpha L)} \\
 &= \frac{1}{L} (1 - \alpha L)^K + \frac{1}{2} \alpha \left(1 - (1 - \alpha L)^K \right). \tag{C.8}
 \end{aligned}$$

Now, if $\alpha \geq \frac{1}{KL}$, then

$$\frac{1}{2} \alpha \left(1 - (1 - \alpha L)^K \right) \geq \frac{\alpha}{8} = \Omega\left(\frac{1}{KL}\right),$$

and hence, $x_{n,K}^2 = \Omega(\frac{1}{K^2 L^2})$. Otherwise, if $\alpha \leq \frac{1}{KL}$, then continuing on from (C.8),

$$\begin{aligned}
 x_{n,K} &\geq \frac{1}{L} (1 - \alpha L)^K + \frac{1}{2} \alpha \left(1 - (1 - \alpha L)^K \right) \\
 &\geq \frac{1}{L} (1 - \alpha L)^K \\
 &\geq \frac{1}{L} e^{-\alpha L K} \\
 &\geq \frac{1}{L} e^{-1} \\
 &= \Omega(1/L),
 \end{aligned}$$

and hence in this case, $x_{n,K}^2 = \Omega(\frac{1}{L^2})$.

C.5 Proof of Theorem 6.9

Let $F(\mathbf{x}) := \frac{1}{n} \sum_{i=1}^n f_i(\mathbf{x})$ be such that its minimizer is at the origin. This can be assumed without loss of generality because we can shift the coordinates appropriately, similar to (Safran and Shamir, 2020). Since the f_i are convex quadratics, we can write them as $f_i(\mathbf{x}) = \frac{1}{2} \mathbf{x}^\top \mathbf{A}_i \mathbf{x} - \mathbf{b}_i^\top \mathbf{x} + c_i$, where $\mathbf{A}_i$ are symmetric, positive-semidefinite matrices. We can omit the constants c_i because they do not affect the minimizer or the gradients. Because we assume that the minimizer of $F(\mathbf{x})$ is at the origin, we get that

$$\sum_{i=1}^n \mathbf{b}_i = \mathbf{0}. \tag{C.9}$$

Let $\sigma = (\sigma_1, \sigma_2, \dots, \sigma_n)$ be the random permutation of $(1, 2, \dots, n)$ generated at the beginning of the algorithm. Then for $k \in (1, 2, \dots, K/2)$, epoch $2k - 1$ sees the n functions in the following sequence:

$$\left(\frac{1}{2} \mathbf{x}^\top \mathbf{A}_{\sigma_1} \mathbf{x} - \mathbf{b}_{\sigma_1}^\top \mathbf{x}, \frac{1}{2} \mathbf{x}^\top \mathbf{A}_{\sigma_2} \mathbf{x} - \mathbf{b}_{\sigma_2}^\top \mathbf{x}, \dots, \frac{1}{2} \mathbf{x}^\top \mathbf{A}_{\sigma_n} \mathbf{x} - \mathbf{b}_{\sigma_n}^\top \mathbf{x} \right),$$

whereas epoch $2k$ sees the n functions in the reverse sequence:

$$\left(\frac{1}{2} \mathbf{x}^\top \mathbf{A}_{\sigma_n} \mathbf{x} - \mathbf{b}_{\sigma_n}^\top \mathbf{x}, \frac{1}{2} \mathbf{x}^\top \mathbf{A}_{\sigma_{n-1}} \mathbf{x} - \mathbf{b}_{\sigma_{n-1}}^\top \mathbf{x}, \dots, \frac{1}{2} \mathbf{x}^\top \mathbf{A}_{\sigma_1} \mathbf{x} - \mathbf{b}_{\sigma_1}^\top \mathbf{x} \right).$$

We define $\mathbf{S}_i := \alpha \mathbf{A}_{\sigma_i}$ and $\mathbf{t}_i = \alpha \mathbf{b}_{\sigma_i}$ for convenience of notation. We start off by computing the progress made during an even indexed epoch. Since the even epochs use the reverse permutation, we get

$$\begin{aligned} \mathbf{x}_0^{2k+1} &= \mathbf{x}_n^{2k} \\ &= \mathbf{x}_{n-1}^{2k} - \alpha (\mathbf{A}_{\sigma_1} \mathbf{x}_{n-1}^{2k} - \mathbf{b}_{\sigma_1}) \quad (f_{\sigma_1} \text{ is used at the last iteration of even epochs.}) \\ &= (\mathbf{I} - \alpha \mathbf{A}_{\sigma_1}) \mathbf{x}_{n-1}^{2k} + \alpha \mathbf{b}_{\sigma_1} \\ &= (\mathbf{I} - \mathbf{S}_1) \mathbf{x}_{n-1}^{2k} + \mathbf{t}_1. \end{aligned}$$

We recursively apply the same procedure as above to the whole epoch to get the following

$$\begin{aligned} \mathbf{x}_0^{2k+1} &= (\mathbf{I} - \mathbf{S}_1) \mathbf{x}_{n-1}^{2k} + \mathbf{t}_1 \\ &= (\mathbf{I} - \mathbf{S}_1) ((\mathbf{I} - \mathbf{S}_2) \mathbf{x}_{n-2}^{2k} + \mathbf{t}_2) + \mathbf{t}_1 \\ &= (\mathbf{I} - \mathbf{S}_1)(\mathbf{I} - \mathbf{S}_2) \mathbf{x}_{n-2}^{2k} + (\mathbf{I} - \mathbf{S}_1) \mathbf{t}_2 + \mathbf{t}_1 \\ &\quad \vdots \\ &= \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i) \right) \mathbf{x}_0^{2k} + \sum_{i=1}^n \left(\prod_{j=1}^{n-i} (\mathbf{I} - \mathbf{S}_j) \right) \mathbf{t}_{n+1-i}, \end{aligned} \tag{C.10}$$

where the product of matrices $\{\mathbf{M}_i\}$ is defined as $\prod_{i=l}^m \mathbf{M}_i = \mathbf{M}_l \mathbf{M}_{l+1} \dots \mathbf{M}_m$ if $m \geq l$ and 1 otherwise. Similar to Eq. (C.10), we can compute the progress made during an odd indexed epoch. Recall that the only difference will be that the odd indexed epochs see the permutations in the order $(\sigma_1, \sigma_2, \dots, \sigma_n)$ instead of $(\sigma_n, \sigma_{n-1}, \dots, \sigma_1)$. After doing the computation, we get the following equation:

$$\mathbf{x}_0^{2k} = \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}) \right) \mathbf{x}_0^{2k-1} + \sum_{i=1}^n \left(\prod_{j=1}^{n-i} (\mathbf{I} - \mathbf{S}_{n-j+1}) \right) \mathbf{t}_i.$$

Combining the results above, we can get the total progress made after the pair of epoch $2k - 1$ and $2k$:

$$\begin{aligned}
\mathbf{x}_0^{2k+1} &= \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i) \right) \mathbf{x}_0^{2k} + \sum_{i=1}^n \left(\prod_{j=1}^{n-i} (\mathbf{I} - \mathbf{S}_j) \right) \mathbf{t}_{n+1-i} \\
&= \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}) \right) \mathbf{x}_0^{2k-1} \\
&\quad + \left(\prod_{j=1}^n (\mathbf{I} - \mathbf{S}_j) \right) \sum_{i=1}^n \left(\prod_{j=1}^{n-i} (\mathbf{I} - \mathbf{S}_{n+1-j}) \right) \mathbf{t}_i + \sum_{i=1}^n \left(\prod_{j=1}^{n-i} (\mathbf{I} - \mathbf{S}_j) \right) \mathbf{t}_{n+1-i}. \quad (\text{C.11})
\end{aligned}$$

In the sum above, the first term will have an exponential decay, hence we need to control the next two terms. We denote the sum of the terms as $\mathbf{z}$ (see the definition below) and we will control its norm later in this proof.

$$\begin{aligned}
\mathbf{z} &:= \left(\prod_{j=1}^n (\mathbf{I} - \mathbf{S}_j) \right) \sum_{i=1}^n \left(\prod_{j=1}^{n-i} (\mathbf{I} - \mathbf{S}_{n+1-j}) \right) \mathbf{t}_i + \sum_{i=1}^n \left(\prod_{j=1}^{n-i} (\mathbf{I} - \mathbf{S}_j) \right) \mathbf{t}_{n+1-i} \\
&= \sum_{i=1}^n \left(\prod_{j=1}^n (\mathbf{I} - \mathbf{S}_j) \right) \left(\prod_{j=1}^{n-i} (\mathbf{I} - \mathbf{S}_{n+1-j}) \right) \mathbf{t}_i + \sum_{i=1}^n \left(\prod_{j=1}^{n-i} (\mathbf{I} - \mathbf{S}_j) \right) \mathbf{t}_{n+1-i}.
\end{aligned}$$

To see where the iterates end up after K epochs, we simply set $2k = K$ in Eq. C.11 and then keep applying the equation recursively to preceding epochs. Then, we get

$$\begin{aligned}
\mathbf{x}_n^K &= \mathbf{x}_0^{K+1} = \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}) \right) \mathbf{x}_0^{K-1} + \mathbf{z} \\
&= \left(\left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}) \right) \right)^2 \mathbf{x}_0^{K-3} \\
&\quad + \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}) \right) \mathbf{z} + \mathbf{z} \\
&\quad \vdots \\
&= \left(\left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}) \right) \right)^{K/2} \mathbf{x}_0^1 \\
&\quad + \sum_{k=0}^{\frac{K}{2}-1} \left(\left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}) \right) \right)^k \mathbf{z}.
\end{aligned}$$

Taking squared norms and expectations on both sides, we get

$$\begin{aligned}
\mathbb{E}[\|\mathbf{x}_n^K\|^2] &= \mathbb{E} \left[\left\| \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}) \right) \right\|^{K/2} \mathbf{x}_0^1 \right. \\
&\quad \left. + \sum_{k=0}^{\frac{K}{2}-1} \left\| \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}) \right) \right\|^k \mathbf{z} \right]^2 \\
&\leq 2\mathbb{E} \left[\left\| \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}) \right) \right\|^{K/2} \mathbf{x}_0^1 \right]^2 \\
&\quad + 2\mathbb{E} \left[\left\| \sum_{k=0}^{\frac{K}{2}-1} \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}) \right) \right\|^k \mathbf{z} \right]^2 \quad (\text{Since } (a+b)^2 \leq 2a^2 + 2b^2) \\
&\leq 2\mathbb{E} \left[\left\| \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}) \right) \right\|^{K/2} \mathbf{x}_0^1 \right]^2 \\
&\quad + 2\mathbb{E} \left[\left(\|\mathbf{z}\| \sum_{k=0}^{\frac{K}{2}-1} \left\| \prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i) \prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}) \right\|^k \right)^2 \right].
\end{aligned}$$

We assumed that the functions f_i have L -Lipschitz gradients (Assumption 6.2). This translates to A_i having maximum eigenvalue less than L . Hence, if $\alpha \leq 1/L$, we get that $\mathbf{I} - \alpha A_i$ is positive semi-definite with maximum eigenvalue bounded by 1. Hence, $\|\mathbf{I} - \mathbf{S}_i\| \leq 1$. Using this and the fact that for matrices $\mathbf{M}_1$ and $\mathbf{M}_2$, $\|\mathbf{M}_1 \mathbf{M}_2\| \leq \|\mathbf{M}_1\| \|\mathbf{M}_2\|$, we get that

$$\begin{aligned}
\mathbb{E}[\|\mathbf{x}_n^K\|^2] &\leq 2\mathbb{E} \left[\left\| \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}) \right) \right\|^{K/2} \mathbf{x}_0^1 \right]^2 \\
&\quad + 2\mathbb{E} \left[\left(\|\mathbf{z}\| \sum_{k=0}^{\frac{K}{2}-1} \left(\prod_{i=1}^n \|\mathbf{I} - \mathbf{S}_i\| \prod_{i=1}^n \|\mathbf{I} - \mathbf{S}_{n-i+1}\| \right)^k \right)^2 \right] \\
&\leq 2\mathbb{E} \left[\left\| \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}) \right) \right\|^{K/2} \mathbf{x}_0^1 \right]^2 + 2\mathbb{E} \left[\left(\|\mathbf{z}\| \sum_{k=0}^{\frac{K}{2}-1} 1 \right)^2 \right] \\
&= 2\mathbb{E} \left[\left\| \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}) \right) \right\|^{K/2} \mathbf{x}_0^1 \right]^2 + \frac{K^2}{2} \mathbb{E}[\|\mathbf{z}\|^2].
\end{aligned}$$

We handle the two terms above separately. For the first term, we have the following bound:

Lemma C.5. *If $\alpha \leq \frac{1}{8\kappa L} \min \left\{ 2, \frac{\sqrt{\kappa}}{n} \right\}$, then*

$$\left\| \prod_{i=1}^n (I - S_i) \prod_{i=1}^n (I - S_{n-i+1}) \right\| \leq 1 - \alpha n \mu$$

Note that $K \geq 80\kappa^{3/2} \log(nK) \max \left\{ 1, \frac{\sqrt{\kappa}}{n} \right\} \implies \alpha \leq \frac{1}{8\kappa L} \min \left\{ 2, \frac{\sqrt{\kappa}}{n} \right\}$.

We also have the following lemma that bounds the expected squared norm of $\mathbf{z}$.

Lemma C.6. *If $\alpha \leq \frac{1}{L}$, then*

$$\mathbb{E} [\|\mathbf{z}\|^2] \leq 2n^2 \alpha^4 L^2 (G^*)^2 + 170n^5 \alpha^6 L^4 G^2 \log n,$$

where $G^* = \max_i \|\mathbf{b}_i\|$, and the expectation is taken over the randomness of $\mathbf{z}$.

Using these lemmas, we get that

$$\begin{aligned} \mathbb{E} [\|\mathbf{x}_n^K\|^2] &\leq 2(1 - n\alpha\mu)^{K/2} \|\mathbf{x}_0^1\|^2 + K^2 n^2 \alpha^4 L^2 G^2 + 85K^2 n^5 \alpha^6 L^4 G^2 \log n \\ &\leq 2e^{-\frac{1}{2}n\alpha\mu K} \|\mathbf{x}_0^1\|^2 + K^2 n^2 \alpha^4 L^2 G^2 + 85K^2 n^5 \alpha^6 L^4 G^2 \log n. \end{aligned}$$

Substituting $\alpha = \frac{10 \log n K}{\mu n K}$ gives us the result.

C.5.1 Proof of Lemma C.5

We define $(\tilde{S}_1, \dots, \tilde{S}_n) := (S_1, \dots, S_n)$ and $(\tilde{S}_{n+1}, \dots, \tilde{S}_{2n}) := (S_n, \dots, S_1)$. Then,

$$\begin{aligned} \left\| \prod_{i=1}^n (I - S_i) \prod_{i=1}^n (I - S_{n-i+1}) \right\| &= \left\| \prod_{i=1}^{2n} (I - \tilde{S}_i) \right\| \\ &= \left\| I - \sum_{i=1}^{2n} \tilde{S}_i + \sum_{i < j} \tilde{S}_i \tilde{S}_j - \dots \right\| \\ &\leq \left\| I - \sum_{i=1}^{2n} \tilde{S}_i + \sum_{i < j} \tilde{S}_i \tilde{S}_j \right\| + \left\| \sum_{i < j < k} \tilde{S}_i \tilde{S}_j \tilde{S}_k \right\| + \dots \\ &\leq \left\| I - \sum_{i=1}^{2n} \tilde{S}_i + \sum_{i < j} \tilde{S}_i \tilde{S}_j \right\| + \sum_{i < j < k} \|\tilde{S}_i\| \|\tilde{S}_j\| \|\tilde{S}_k\| + \dots \end{aligned}$$

Note that

$$\sum_{i < j} \tilde{S}_i \tilde{S}_j = 2 \sum_{i \neq j} S_i S_j + \sum_{i=1}^n S_i^2$$

$$= 2 \left(\sum_{i=1}^n \mathbf{S}_i \right) \left(\sum_{i=1}^n \mathbf{S}_i \right) - \sum_{i=1}^n \mathbf{S}_i^2.$$

Substituting this and noting that $\sum_{i=1}^{2n} \tilde{\mathbf{S}}_i = 2 \sum_{i=1}^n \mathbf{S}_i$, we get

$$\begin{aligned} \left\| \prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i) \prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}) \right\| &\leq \left\| \mathbf{I} - 2 \sum_{i=1}^n \mathbf{S}_i + 2 \left(\sum_{i=1}^n \mathbf{S}_i \right) \left(\sum_{i=1}^n \mathbf{S}_i \right) - \sum_{i=1}^n \mathbf{S}_i^2 \right\| \\ &\quad + \sum_{i < j < k} \|\tilde{\mathbf{S}}_i\| \|\tilde{\mathbf{S}}_j\| \|\tilde{\mathbf{S}}_k\| + \dots \\ &\leq \left\| \mathbf{I} - 2 \sum_{i=1}^n \mathbf{S}_i + 2 \left(\sum_{i=1}^n \mathbf{S}_i \right) \left(\sum_{i=1}^n \mathbf{S}_i \right) \right\| + \left\| \sum_{i=1}^n \mathbf{S}_i^2 \right\| \\ &\quad + \sum_{i < j < k} \|\tilde{\mathbf{S}}_i\| \|\tilde{\mathbf{S}}_j\| \|\tilde{\mathbf{S}}_k\| + \dots \end{aligned}$$

Let us denote $\mathbf{T} := \sum_{i=1}^n \mathbf{S}_i$. Then, we know by Assumptions 6.2 and 6.3 that $\mathbf{T}$ has eigenvalues in $[n\alpha\mu, n\alpha L]$. Then, as long as $\alpha \leq \frac{1}{4nL}$, we get that

$$\begin{aligned} \left\| \mathbf{I} - 2 \sum_{i=1}^n \mathbf{S}_i + 2 \left(\sum_{i=1}^n \mathbf{S}_i \right) \left(\sum_{i=1}^n \mathbf{S}_i \right) \right\| &= \left\| \mathbf{I} - 2\mathbf{T} + 2\mathbf{T}^2 \right\| \\ &\leq 1 - \frac{3}{2}n\alpha\mu. \end{aligned}$$

Substituting this, we get

$$\begin{aligned} \left\| \prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i) \prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}) \right\| &\leq \left(1 - \frac{3}{2}n\alpha\mu \right) + \left\| \sum_{i=1}^n \mathbf{S}_i^2 \right\| \\ &\quad + \sum_{i < j < k} \|\tilde{\mathbf{S}}_i\| \|\tilde{\mathbf{S}}_j\| \|\tilde{\mathbf{S}}_k\| + \dots \end{aligned}$$

By Assumption 6.2, we know that $\|\mathbf{A}_i\| \leq L$. Hence, $\|\tilde{\mathbf{S}}_i\| \leq \alpha L$. Hence,

$$\begin{aligned} \left\| \prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i) \prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}) \right\| &\leq \left(1 - \frac{3}{2}n\alpha\mu \right) + n\alpha^2 L^2 \\ &\quad + \left(\binom{2n}{3} \alpha^3 L^3 + \binom{2n}{4} \alpha^4 L^4 + \dots \right) \\ &\leq 1 - \frac{3}{2}n\alpha\mu + n\alpha^2 L^2 + \sum_{i=3}^{2n} (2n\alpha L)^i \\ &\leq 1 - \frac{3}{2}n\alpha\mu + n\alpha^2 L^2 + \frac{8n^3 \alpha^3 L^3}{1 - 2n\alpha L} \\ &\leq 1 - \frac{3}{2}n\alpha\mu + n\alpha^2 L^2 + 16n^3 \alpha^3 L^3. \quad (\text{Since } \alpha \leq 1/4nL.) \end{aligned}$$

Finally, as long as $\alpha \leq \frac{1}{4\kappa L}$ and $\alpha \leq \frac{1}{8nL\sqrt{\kappa}}$,

$$\left\| \prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i) \prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}) \right\| \leq 1 - n\alpha\mu.$$

C.5.2 Proof of Lemma C.6

We start off by computing the first order expansion of $\mathbf{z}$. We have the following lemma for this:

Lemma C.7.

$$\begin{aligned} \mathbf{z} = & \sum_{i=1}^n \mathbf{S}_i \mathbf{t}_i + \sum_{j=n+1}^{2n-1} \sum_{l=1}^{j-1} \left(\prod_{p=1}^{l-1} (\mathbf{I} - \tilde{\mathbf{S}}_p) \right) \tilde{\mathbf{S}}_l \tilde{\mathbf{S}}_j \left(\sum_{i=1}^{2n-j} \mathbf{t}_i \right) \\ & + \sum_{j=1}^{n-1} \sum_{l=1}^{j-1} \left(\prod_{p=1}^{l-1} (\mathbf{I} - \mathbf{S}_p) \right) \mathbf{S}_l \mathbf{S}_j \left(\sum_{i=1}^{n-j} \mathbf{t}_{n+1-i} \right), \end{aligned}$$

where $(\tilde{\mathbf{S}}_1, \dots, \tilde{\mathbf{S}}_n) := (\mathbf{S}_1, \dots, \mathbf{S}_n)$ and $(\tilde{\mathbf{S}}_{n+1}, \dots, \tilde{\mathbf{S}}_{2n}) := (\mathbf{S}_n, \dots, \mathbf{S}_1)$.

The proof of this lemma is quite algebraic and hence has been pushed to the end, in Appendix C.5.4.

The strategy is to bound $\|\mathbf{S}_i \mathbf{t}_i\|$, $\|\sum_{i=1}^{2n-j} \mathbf{t}_i\|$, and $\|\sum_{i=1}^{n-j} \mathbf{t}_{n+1-i}\|$. Hence, we apply Lemma C.7 and use triangle inequality:

$$\begin{aligned} \mathbb{E} [\|\mathbf{z}\|^2] &= \mathbb{E} \left[\left\| \sum_{i=1}^n \mathbf{S}_i \mathbf{t}_i + \sum_{j=n+1}^{2n-1} \sum_{l=1}^{j-1} \left(\prod_{p=1}^{l-1} (\mathbf{I} - \tilde{\mathbf{S}}_p) \right) \tilde{\mathbf{S}}_l \tilde{\mathbf{S}}_j \left(\sum_{i=1}^{2n-j} \mathbf{t}_i \right) \right. \right. \\ &\quad \left. \left. + \sum_{j=1}^{n-1} \sum_{l=1}^{j-1} \left(\prod_{p=1}^{l-1} (\mathbf{I} - \mathbf{S}_p) \right) \mathbf{S}_l \mathbf{S}_j \left(\sum_{i=1}^{n-j} \mathbf{t}_{n+1-i} \right) \right\|^2 \right] \\ &\leq \mathbb{E} \left[\left(\sum_{i=1}^n \|\mathbf{S}_i\| \|\mathbf{t}_i\| + \sum_{j=n+1}^{2n-1} \sum_{l=1}^{j-1} \left(\prod_{p=1}^{l-1} \|\mathbf{I} - \tilde{\mathbf{S}}_p\| \right) \|\tilde{\mathbf{S}}_l\| \|\tilde{\mathbf{S}}_j\| \left(\sum_{i=1}^{2n-j} \|\mathbf{t}_i\| \right) \right. \right. \\ &\quad \left. \left. + \sum_{j=1}^{n-1} \sum_{l=1}^{j-1} \left(\prod_{p=1}^{l-1} \|\mathbf{I} - \mathbf{S}_p\| \right) \|\mathbf{S}_l\| \|\mathbf{S}_j\| \left(\sum_{i=1}^{n-j} \|\mathbf{t}_{n+1-i}\| \right) \right)^2 \right]. \end{aligned}$$

Now, we recall that $\|\mathbf{S}_i\| \leq \alpha L$ and $\|\mathbf{t}_i\| \leq \alpha G$. Because $\alpha \leq 1/L$, we also get that $\|\mathbf{I} - \mathbf{S}_i\| \leq 1$. Using these,

$$\mathbb{E} [\|\mathbf{z}\|^2] \leq \mathbb{E} \left[\left(\sum_{i=1}^n \alpha^2 L G + \sum_{j=n+1}^{2n-1} \sum_{l=1}^{j-1} \left(\prod_{p=1}^{l-1} 1 \right) \alpha L \alpha L \left\| \sum_{i=1}^{2n-j} \mathbf{t}_i \right\| \right. \right.$$

$$\begin{aligned}
& + \sum_{j=1}^{n-1} \sum_{l=1}^{j-1} \left(\prod_{p=1}^{l-1} 1 \right) \alpha L \alpha L \left\| \sum_{i=1}^{n-j} \mathbf{t}_{n+1-i} \right\|^2 \Bigg] \\
& \leq \mathbb{E} \left[\left(n\alpha^2 L G + 2n\alpha^2 L^2 \sum_{j=n+1}^{2n-1} \left\| \sum_{i=1}^{2n-j} \mathbf{t}_i \right\|^2 + n\alpha^2 L^2 \sum_{j=1}^{n-1} \left\| \sum_{i=1}^{n-j} \mathbf{t}_{n+1-i} \right\|^2 \right)^2 \right] \\
& = n^2 \alpha^4 L^2 \mathbb{E} \left[\left(G + 2L \sum_{j=n+1}^{2n-1} \left\| \sum_{i=1}^{2n-j} \mathbf{t}_i \right\|^2 + L \sum_{j=1}^{n-1} \left\| \sum_{i=1}^{n-j} \mathbf{t}_{n+1-i} \right\|^2 \right)^2 \right] \\
& \leq 2n^2 \alpha^4 L^2 \left(G^2 + L^2 \mathbb{E} \left[\left(2 \sum_{j=n+1}^{2n-1} \left\| \sum_{i=1}^{2n-j} \mathbf{t}_i \right\|^2 + \sum_{j=1}^{n-1} \left\| \sum_{i=1}^{n-j} \mathbf{t}_{n+1-i} \right\|^2 \right)^2 \right] \right). \\
& \hspace{25em} (\text{Since } (a+b)^2 \leq 2a^2 + 2b^2)
\end{aligned}$$

Using Hoeffding-Serfling inequality for bounded random vectors (Schneider, 2016, Theorem 2), we get the following lemma

Lemma C.8. *For all $j, l \in [1, n]$ we have that*

$$\begin{aligned}
\mathbb{E} \left[\left\| \sum_{i=1}^j \mathbf{t}_i \right\|^2 \right] & \leq 18j\alpha^2 (G^*)^2 \log(n) \\
\mathbb{E} \left[\left\| \sum_{i=1}^j \mathbf{t}_i \right\| \left\| \sum_{i=1}^l \mathbf{t}_i \right\| \right] & \leq 18\sqrt{jl}\alpha^2 (G^*)^2 \log(n),
\end{aligned}$$

where $G^* = \max_i \|\mathbf{b}_i\|$, and the expectation is taken over the randomness of $\mathbf{t}_i$.

Writing out the expansion of $\left(2 \sum_{j=n+1}^{2n-1} \left\| \sum_{i=1}^{2n-j} \mathbf{t}_i \right\|^2 + \sum_{j=1}^{n-1} \left\| \sum_{i=1}^{n-j} \mathbf{t}_{n+1-i} \right\|^2 \right)^2$ and using the lemma above on the individual terms, we get

$$\mathbb{E} \left[\|\mathbf{z}\|^2 \right] \leq 2n^2 \alpha^4 L^2 G^2 + 2n^2 \alpha^4 L^4 (90\alpha^2 n^3 G^2 \log n).$$

C.5.3 Proof of Lemma C.8

This proof is similar to the one in (Ahn et al., 2020). Define $G^* := \max_i \|\mathbf{b}_i\|$. We use the following theorem adapted to our setting

Theorem C.9. [(Schneider, 2016, Theorem 2)] *With probability at least $1 - \frac{\delta}{n}$,*

$$\left\| \sum_{i=1}^j \mathbf{t}_i \right\| \leq \alpha G^* \sqrt{8j \left(1 - \frac{j-1}{n} \right) \log \frac{2n}{\delta}}.$$

Then taking a union bound over $j = 1, \dots, n$, we get that with probability at least $1 - \delta$,

$$\forall j \in [1, n] : \left\| \sum_{i=1}^j \mathbf{t}_i \right\| \leq \alpha G^* \sqrt{8j \left(1 - \frac{j-1}{n}\right) \log \frac{2n}{\delta}} \leq \alpha G^* \sqrt{8j \log \frac{2n}{\delta}}.$$

Then, for the complementary event (which happens with probability δ), we use the fact that $\|\mathbf{t}_i\| = \|\alpha \mathbf{b}_{\sigma_i}\| \leq \alpha G^*$ to get the following:

$$\forall j \in [1, n] : \left\| \sum_{i=1}^j \mathbf{t}_i \right\| \leq \sum_{i=1}^j \|\mathbf{t}_i\| \leq \alpha G^* j.$$

Now, choose $\delta = 1/n$. Then, we get that

$$\begin{aligned} \mathbb{E} \left[\left\| \sum_{i=1}^j \mathbf{t}_i \right\|^2 \right] &\leq \left(1 - \frac{1}{n}\right) 8j\alpha^2 G^2 \log(2n^2) + \frac{1}{n} (\alpha G^* j)^2 \\ &\leq 18j\alpha^2 (G^*)^2 \log n. \end{aligned}$$

Similarly, we can also get

$$\mathbb{E} \left[\left\| \sum_{i=1}^j \mathbf{t}_i \right\| \left\| \sum_{i=1}^l \mathbf{t}_i \right\| \right] \leq 18\sqrt{jl}\alpha^2 (G^*)^2 \log(n).$$

C.5.4 Proof of Lemma C.7

As in the lemma's statement, define $(\tilde{\mathbf{S}}_1, \dots, \tilde{\mathbf{S}}_{2n})$ as $(\tilde{\mathbf{S}}_1, \dots, \tilde{\mathbf{S}}_n) = (\mathbf{S}_1, \dots, \mathbf{S}_n)$ and $(\tilde{\mathbf{S}}_{n+1}, \dots, \tilde{\mathbf{S}}_{2n}) = (\mathbf{S}_n, \dots, \mathbf{S}_1)$. As a reminder, the term $\mathbf{z}$ is defined as follows:

$$\mathbf{z} := \sum_{i=1}^n \left(\prod_{j=1}^n (\mathbf{I} - \mathbf{S}_j) \right) \left(\prod_{j=1}^{n-i} (\mathbf{I} - \mathbf{S}_{n+1-j}) \right) \mathbf{t}_i + \sum_{i=1}^n \left(\prod_{j=1}^{n-i} (\mathbf{I} - \mathbf{S}_j) \right) \mathbf{t}_{n+1-i}. \quad (\text{C.12})$$

First, we analyze the first term in $\mathbf{z}$. Towards that end, we start by expanding $\left(\prod_{j=1}^n (\mathbf{I} - \mathbf{S}_j) \right) \left(\prod_{j=1}^{n-i} (\mathbf{I} - \mathbf{S}_{n+1-j}) \right)$

$$\begin{aligned} \left(\prod_{j=1}^n (\mathbf{I} - \mathbf{S}_j) \right) \left(\prod_{j=1}^{n-i} (\mathbf{I} - \mathbf{S}_{n+1-j}) \right) &= \left(\prod_{j=1}^{2n-i} (\mathbf{I} - \tilde{\mathbf{S}}_j) \right) \\ &= \left(\prod_{j=1}^{2n-i-1} (\mathbf{I} - \tilde{\mathbf{S}}_j) \right) (\mathbf{I} - \tilde{\mathbf{S}}_{2n-i}) \\ &= \left(\prod_{j=1}^{2n-i-1} (\mathbf{I} - \tilde{\mathbf{S}}_j) \right) - \left(\prod_{j=1}^{2n-i-1} (\mathbf{I} - \tilde{\mathbf{S}}_j) \right) \tilde{\mathbf{S}}_{2n-i}. \end{aligned} \quad (\text{C.13})$$

Similarly, we expand the term $\left(\prod_{j=1}^{2n-i-1}(\mathbf{I} - \tilde{\mathbf{S}}_j)\right)$ and then recursively keep doing it to get the following:

$$\begin{aligned}
\left(\prod_{j=1}^n(\mathbf{I} - \mathbf{S}_j)\right) \left(\prod_{j=1}^{n-i}(\mathbf{I} - \mathbf{S}_{n+1-j})\right) &= \left(\prod_{j=1}^{2n-i-1}(\mathbf{I} - \tilde{\mathbf{S}}_j)\right) - \left(\prod_{j=1}^{2n-i-1}(\mathbf{I} - \tilde{\mathbf{S}}_j)\right) \tilde{\mathbf{S}}_{2n-i} \\
&= \left(\prod_{j=1}^{2n-i-2}(\mathbf{I} - \tilde{\mathbf{S}}_j)\right) - \left(\prod_{j=1}^{2n-i-2}(\mathbf{I} - \tilde{\mathbf{S}}_j)\right) \tilde{\mathbf{S}}_{2n-i-1} \\
&\quad - \left(\prod_{j=1}^{2n-i-1}(\mathbf{I} - \tilde{\mathbf{S}}_j)\right) \tilde{\mathbf{S}}_{2n-i} \\
&= \vdots \\
&= (\mathbf{I} - \tilde{\mathbf{S}}_1) - (\mathbf{I} - \tilde{\mathbf{S}}_1) \tilde{\mathbf{S}}_2 - \dots \\
&\quad - \left(\prod_{j=1}^{2n-i-2}(\mathbf{I} - \tilde{\mathbf{S}}_j)\right) \tilde{\mathbf{S}}_{2n-i-1} - \left(\prod_{j=1}^{2n-i-1}(\mathbf{I} - \tilde{\mathbf{S}}_j)\right) \tilde{\mathbf{S}}_{2n-i} \\
&= \mathbf{I} - \sum_{j=1}^{2n-i} \left(\prod_{l=1}^{j-1}(\mathbf{I} - \tilde{\mathbf{S}}_l)\right) \tilde{\mathbf{S}}_j.
\end{aligned}$$

Note that the term $\prod_{l=1}^{j-1}(\mathbf{I} - \tilde{\mathbf{S}}_l)$ above is similar to the RHS of Eq. (C.13). Hence, we repeat the process again on this term to get the following

$$\begin{aligned}
\left(\prod_{j=1}^n(\mathbf{I} - \mathbf{S}_j)\right) \left(\prod_{j=1}^{n-i}(\mathbf{I} - \mathbf{S}_{n+1-j})\right) &= \mathbf{I} - \sum_{j=1}^{2n-i} \left(\mathbf{I} - \sum_{l=1}^{j-1} \left(\prod_{p=1}^{l-1}(\mathbf{I} - \tilde{\mathbf{S}}_p)\right) \tilde{\mathbf{S}}_l\right) \tilde{\mathbf{S}}_j \\
&= \mathbf{I} - \sum_{j=1}^{2n-i} \tilde{\mathbf{S}}_j + \sum_{j=1}^{2n-i} \sum_{l=1}^{j-1} \left(\prod_{p=1}^{l-1}(\mathbf{I} - \tilde{\mathbf{S}}_p)\right) \tilde{\mathbf{S}}_l \tilde{\mathbf{S}}_j. \quad (\text{C.14})
\end{aligned}$$

Using this in the first term $\sum_{i=1}^n \left(\prod_{j=1}^n(\mathbf{I} - \mathbf{S}_j)\right) \left(\prod_{j=1}^{n-i}(\mathbf{I} - \mathbf{S}_{n+1-j})\right) \mathbf{t}_i$ (in Eq. (C.12)), we get

$$\begin{aligned}
\sum_{i=1}^n \left(\prod_{j=1}^n(\mathbf{I} - \mathbf{S}_j)\right) \left(\prod_{j=1}^{n-i}(\mathbf{I} - \mathbf{S}_{n+1-j})\right) \mathbf{t}_i &= \sum_{i=1}^n \left(\mathbf{I} - \sum_{j=1}^{2n-i} \tilde{\mathbf{S}}_j + \sum_{j=1}^{2n-i} \sum_{l=1}^{j-1} \left(\prod_{p=1}^{l-1}(\mathbf{I} - \tilde{\mathbf{S}}_p)\right) \tilde{\mathbf{S}}_l \tilde{\mathbf{S}}_j\right) \mathbf{t}_i \\
&= \sum_{i=1}^n \mathbf{t}_i - \sum_{i=1}^n \sum_{j=1}^{2n-i} \tilde{\mathbf{S}}_j \mathbf{t}_i + \sum_{i=1}^n \sum_{j=1}^{2n-i} \sum_{l=1}^{j-1} \left(\prod_{p=1}^{l-1}(\mathbf{I} - \tilde{\mathbf{S}}_p)\right) \tilde{\mathbf{S}}_l \tilde{\mathbf{S}}_j \mathbf{t}_i.
\end{aligned}$$

Now, we use the fact that $\sum_{i=1}^n \mathbf{b}_i = \mathbf{0}$ (Eq. (C.9)) to get that $\sum_{i=1}^n \mathbf{t}_i = \mathbf{0}$. Then,

$$\begin{aligned} & \sum_{i=1}^n \left(\prod_{j=1}^n (\mathbf{I} - \mathbf{S}_j) \right) \left(\prod_{j=1}^{n-i} (\mathbf{I} - \mathbf{S}_{n+1-j}) \right) \mathbf{t}_i \\ &= - \sum_{i=1}^n \sum_{j=1}^{2n-i} \tilde{\mathbf{S}}_j \mathbf{t}_i + \sum_{i=1}^n \sum_{j=1}^{2n-i} \sum_{l=1}^{j-1} \left(\prod_{p=1}^{l-1} (\mathbf{I} - \tilde{\mathbf{S}}_p) \right) \tilde{\mathbf{S}}_l \tilde{\mathbf{S}}_j \mathbf{t}_i \end{aligned}$$

For convenience, we define $\widetilde{\mathbf{M}}_j := \sum_{l=1}^{j-1} \left(\prod_{p=1}^{l-1} (\mathbf{I} - \tilde{\mathbf{S}}_p) \right) \tilde{\mathbf{S}}_l \tilde{\mathbf{S}}_j$. Then,

$$\begin{aligned} & \sum_{i=1}^n \left(\prod_{j=1}^n (\mathbf{I} - \mathbf{S}_j) \right) \left(\prod_{j=1}^{n-i} (\mathbf{I} - \mathbf{S}_{n+1-j}) \right) \mathbf{t}_i = - \sum_{i=1}^n \sum_{j=1}^{2n-i} \tilde{\mathbf{S}}_j \mathbf{t}_i + \sum_{i=1}^n \sum_{j=1}^{2n-i} \widetilde{\mathbf{M}}_j \mathbf{t}_i \\ &= - \sum_{i=1}^n \sum_{j=1}^{2n-i} \tilde{\mathbf{S}}_j \mathbf{t}_i + \sum_{i=1}^n \sum_{j=1}^n \widetilde{\mathbf{M}}_j \mathbf{t}_i + \sum_{i=1}^n \sum_{j=n+1}^{2n-i} \widetilde{\mathbf{M}}_j \mathbf{t}_i \\ &= - \sum_{i=1}^n \sum_{j=1}^{2n-i} \tilde{\mathbf{S}}_j \mathbf{t}_i + \sum_{j=1}^n \widetilde{\mathbf{M}}_j \sum_{i=1}^n \mathbf{t}_i + \sum_{i=1}^n \sum_{j=n+1}^{2n-i} \widetilde{\mathbf{M}}_j \mathbf{t}_i \\ &= - \sum_{i=1}^n \sum_{j=1}^{2n-i} \tilde{\mathbf{S}}_j \mathbf{t}_i + \sum_{i=1}^n \sum_{j=n+1}^{2n-i} \widetilde{\mathbf{M}}_j \mathbf{t}_i. \quad (\text{Since } \sum_{i=1}^n \mathbf{t}_i = \mathbf{0}) \end{aligned}$$

Note that $\sum_{i=1}^n \sum_{j=n+1}^{2n-i} \widetilde{\mathbf{M}}_j \mathbf{t}_i = \sum_{j=n+1}^{2n-1} \widetilde{\mathbf{M}}_j \left(\sum_{i=1}^{2n-j} \mathbf{t}_i \right)$. Hence,

$$\begin{aligned} & \sum_{i=1}^n \left(\prod_{j=1}^n (\mathbf{I} - \mathbf{S}_j) \right) \left(\prod_{j=1}^{n-i} (\mathbf{I} - \mathbf{S}_{n+1-j}) \right) \mathbf{t}_i = - \sum_{i=1}^n \sum_{j=1}^{2n-i} \tilde{\mathbf{S}}_j \mathbf{t}_i + \sum_{j=n+1}^{2n-1} \widetilde{\mathbf{M}}_j \left(\sum_{i=1}^{2n-j} \mathbf{t}_i \right) \\ &= - \sum_{i=1}^n \sum_{j=1}^n \tilde{\mathbf{S}}_j \mathbf{t}_i - \sum_{i=1}^n \sum_{j=n+1}^{2n-i} \tilde{\mathbf{S}}_j \mathbf{t}_i + \sum_{j=n+1}^{2n-1} \widetilde{\mathbf{M}}_j \left(\sum_{i=1}^{2n-j} \mathbf{t}_i \right) \\ &= - \sum_{j=1}^n \tilde{\mathbf{S}}_j \sum_{i=1}^n \mathbf{t}_i - \sum_{i=1}^n \sum_{j=n+1}^{2n-i} \tilde{\mathbf{S}}_j \mathbf{t}_i + \sum_{j=n+1}^{2n-1} \widetilde{\mathbf{M}}_j \left(\sum_{i=1}^{2n-j} \mathbf{t}_i \right) \\ &= - \sum_{i=1}^n \sum_{j=n+1}^{2n-i} \tilde{\mathbf{S}}_j \mathbf{t}_i + \sum_{j=n+1}^{2n-1} \widetilde{\mathbf{M}}_j \left(\sum_{i=1}^{2n-j} \mathbf{t}_i \right) \quad (\text{Since } \sum_{i=1}^n \mathbf{t}_i = \mathbf{0}) \\ &= - \sum_{i=1}^n \sum_{j=n+1}^{2n-i} \tilde{\mathbf{S}}_j \mathbf{t}_i + \sum_{j=n+1}^{2n-1} \sum_{l=1}^{j-1} \left(\prod_{p=1}^{l-1} (\mathbf{I} - \tilde{\mathbf{S}}_p) \right) \tilde{\mathbf{S}}_l \tilde{\mathbf{S}}_j \left(\sum_{i=1}^{2n-j} \mathbf{t}_i \right) \\ &= - \sum_{i=1}^n \sum_{j=i+1}^n \mathbf{S}_j \mathbf{t}_i + \sum_{j=n+1}^{2n-1} \sum_{l=1}^{j-1} \left(\prod_{p=1}^{l-1} (\mathbf{I} - \tilde{\mathbf{S}}_p) \right) \tilde{\mathbf{S}}_l \tilde{\mathbf{S}}_j \left(\sum_{i=1}^{2n-j} \mathbf{t}_i \right). \quad (\text{C.15}) \end{aligned}$$

Next we analyze the second term in $\mathbf{z}$. For this, we start by expanding $\prod_{j=1}^{n-i} (\mathbf{I} - \mathbf{S}_j)$ in

a similar way as Eq. (C.14)

$$\prod_{j=1}^{n-i} (\mathbf{I} - \mathbf{S}_j) = \mathbf{I} - \sum_{j=1}^{n-i} \mathbf{S}_j + \sum_{j=1}^{n-i} \sum_{l=1}^{j-1} \left(\prod_{p=1}^{l-1} (\mathbf{I} - \mathbf{S}_p) \right) \mathbf{S}_l \mathbf{S}_j \quad (\text{C.16})$$

Using this, we get

$$\begin{aligned} \sum_{i=1}^n \left(\prod_{j=1}^{n-i} (\mathbf{I} - \mathbf{S}_j) \right) \mathbf{t}_{n+1-i} &= \sum_{i=1}^n \left(\mathbf{I} - \sum_{j=1}^{n-i} \mathbf{S}_j + \sum_{j=1}^{n-i} \sum_{l=1}^{j-1} \left(\prod_{p=1}^{l-1} (\mathbf{I} - \mathbf{S}_p) \right) \mathbf{S}_l \mathbf{S}_j \right) \mathbf{t}_{n+1-i} \\ &= \sum_{i=1}^n \mathbf{t}_{n+1-i} - \sum_{i=1}^n \sum_{j=1}^{n-i} \mathbf{S}_j \mathbf{t}_{n+1-i} + \sum_{i=1}^n \sum_{j=1}^{n-i} \sum_{l=1}^{j-1} \left(\prod_{p=1}^{l-1} (\mathbf{I} - \mathbf{S}_p) \right) \mathbf{S}_l \mathbf{S}_j \mathbf{t}_{n+1-i} \\ &= - \sum_{i=1}^n \sum_{j=1}^{n-i} \mathbf{S}_j \mathbf{t}_{n+1-i} + \sum_{i=1}^n \sum_{j=1}^{n-i} \sum_{l=1}^{j-1} \left(\prod_{p=1}^{l-1} (\mathbf{I} - \mathbf{S}_p) \right) \mathbf{S}_l \mathbf{S}_j \mathbf{t}_{n+1-i}, \end{aligned}$$

where we used the fact that $\sum_{i=1}^n \mathbf{t}_i = \mathbf{0}$ in the last equality. For convenience, we define $\mathbf{M}_j := \sum_{l=1}^{j-1} \left(\prod_{p=1}^{l-1} (\mathbf{I} - \mathbf{S}_p) \right) \mathbf{S}_l \mathbf{S}_j$. Then,

$$\sum_{i=1}^n \left(\prod_{j=1}^{n-i} (\mathbf{I} - \mathbf{S}_j) \right) \mathbf{t}_{n+1-i} = - \sum_{i=1}^n \sum_{j=1}^{n-i} \mathbf{S}_j \mathbf{t}_{n+1-i} + \sum_{i=1}^n \sum_{j=1}^{n-i} \mathbf{M}_j \mathbf{t}_{n+1-i}.$$

Since $\sum_{i=1}^n \sum_{j=1}^{n-i} \mathbf{M}_j \mathbf{t}_{n+1-i} = \sum_{j=1}^{n-1} \mathbf{M}_j \left(\sum_{i=1}^{n-j} \mathbf{t}_{n+1-i} \right)$, we get

$$\begin{aligned} \sum_{i=1}^n \left(\prod_{j=1}^{n-i} (\mathbf{I} - \mathbf{S}_j) \right) \mathbf{t}_{n+1-i} &= - \sum_{i=1}^n \sum_{j=1}^{n-i} \mathbf{S}_j \mathbf{t}_{n+1-i} + \sum_{j=1}^{n-1} \mathbf{M}_j \left(\sum_{i=1}^{n-j} \mathbf{t}_{n+1-i} \right) \\ &= - \sum_{l=1}^n \sum_{j=1}^{l-1} \mathbf{S}_j \mathbf{t}_l + \sum_{j=1}^{n-1} \mathbf{M}_j \left(\sum_{i=1}^{n-j} \mathbf{t}_{n+1-i} \right) \\ &= - \sum_{i=1}^n \sum_{j=1}^{i-1} \mathbf{S}_j \mathbf{t}_i + \sum_{j=1}^{n-1} \mathbf{M}_j \left(\sum_{i=1}^{n-j} \mathbf{t}_{n+1-i} \right) \\ &= - \sum_{i=1}^n \sum_{j=1}^{i-1} \mathbf{S}_j \mathbf{t}_i + \sum_{j=1}^{n-1} \sum_{l=1}^{j-1} \left(\prod_{p=1}^{l-1} (\mathbf{I} - \mathbf{S}_p) \right) \mathbf{S}_l \mathbf{S}_j \left(\sum_{i=1}^{n-j} \mathbf{t}_{n+1-i} \right). \quad (\text{C.17}) \end{aligned}$$

Finally, substituting Eq. (C.15) and (C.17) in the definition of $\mathbf{z}$ (Eq. (C.12)), we get

$$\begin{aligned} \mathbf{z} &= - \sum_{i=1}^n \sum_{j=i+1}^n \mathbf{S}_j \mathbf{t}_i + \sum_{j=n+1}^{2n-1} \sum_{l=1}^{j-1} \left(\prod_{p=1}^{l-1} (\mathbf{I} - \tilde{\mathbf{S}}_p) \right) \tilde{\mathbf{S}}_l \tilde{\mathbf{S}}_j \left(\sum_{i=1}^{2n-j} \mathbf{t}_i \right) - \sum_{i=1}^n \sum_{j=1}^{i-1} \mathbf{S}_j \mathbf{t}_i \\ &\quad + \sum_{j=1}^{n-1} \sum_{l=1}^{j-1} \left(\prod_{p=1}^{l-1} (\mathbf{I} - \mathbf{S}_p) \right) \mathbf{S}_l \mathbf{S}_j \left(\sum_{i=1}^{n-j} \mathbf{t}_{n+1-i} \right) \end{aligned}$$

$$\begin{aligned}
&= - \sum_{i=1}^n \left(\sum_{j=1}^{i-1} \mathbf{S}_j + \sum_{j=i+1}^n \mathbf{S}_j \right) \mathbf{t}_i + \sum_{j=n+1}^{2n-1} \sum_{l=1}^{j-1} \left(\prod_{p=1}^{l-1} (\mathbf{I} - \tilde{\mathbf{S}}_p) \right) \tilde{\mathbf{S}}_l \tilde{\mathbf{S}}_j \left(\sum_{i=1}^{2n-j} \mathbf{t}_i \right) \\
&\quad + \sum_{j=1}^{n-1} \sum_{l=1}^{j-1} \left(\prod_{p=1}^{l-1} (\mathbf{I} - \mathbf{S}_p) \right) \mathbf{S}_l \mathbf{S}_j \left(\sum_{i=1}^{n-j} \mathbf{t}_{n+1-i} \right) \\
&= - \sum_{i=1}^n \left(\sum_{j=1}^n \mathbf{S}_j \right) \mathbf{t}_i + \sum_{i=1}^n \mathbf{S}_i \mathbf{t}_i + \sum_{j=n+1}^{2n-1} \sum_{l=1}^{j-1} \left(\prod_{p=1}^{l-1} (\mathbf{I} - \tilde{\mathbf{S}}_p) \right) \tilde{\mathbf{S}}_l \tilde{\mathbf{S}}_j \left(\sum_{i=1}^{2n-j} \mathbf{t}_i \right) \\
&\quad + \sum_{j=1}^{n-1} \sum_{l=1}^{j-1} \left(\prod_{p=1}^{l-1} (\mathbf{I} - \mathbf{S}_p) \right) \mathbf{S}_l \mathbf{S}_j \left(\sum_{i=1}^{n-j} \mathbf{t}_{n+1-i} \right) \\
&= - \left(\sum_{j=1}^n \mathbf{S}_j \right) \sum_{i=1}^n \mathbf{t}_i + \sum_{i=1}^n \mathbf{S}_i \mathbf{t}_i + \sum_{j=n+1}^{2n-1} \sum_{l=1}^{j-1} \left(\prod_{p=1}^{l-1} (\mathbf{I} - \tilde{\mathbf{S}}_p) \right) \tilde{\mathbf{S}}_l \tilde{\mathbf{S}}_j \left(\sum_{i=1}^{2n-j} \mathbf{t}_i \right) \\
&\quad + \sum_{j=1}^{n-1} \sum_{l=1}^{j-1} \left(\prod_{p=1}^{l-1} (\mathbf{I} - \mathbf{S}_p) \right) \mathbf{S}_l \mathbf{S}_j \left(\sum_{i=1}^{n-j} \mathbf{t}_{n+1-i} \right) \\
&= \sum_{i=1}^n \mathbf{S}_i \mathbf{t}_i + \sum_{j=n+1}^{2n-1} \sum_{l=1}^{j-1} \left(\prod_{p=1}^{l-1} (\mathbf{I} - \tilde{\mathbf{S}}_p) \right) \tilde{\mathbf{S}}_l \tilde{\mathbf{S}}_j \left(\sum_{i=1}^{2n-j} \mathbf{t}_i \right) \\
&\quad + \sum_{j=1}^{n-1} \sum_{l=1}^{j-1} \left(\prod_{p=1}^{l-1} (\mathbf{I} - \mathbf{S}_p) \right) \mathbf{S}_l \mathbf{S}_j \left(\sum_{i=1}^{n-j} \mathbf{t}_{n+1-i} \right),
\end{aligned}$$

where we used the fact that $\sum_{i=1}^n \mathbf{t}_i = \mathbf{0}$ in the last equality.

C.6 Proof of Theorem 6.10

The proof is similar to that of Theorem 6.9, except for that here we leverage the independence of random permutations in every other epoch. The setup is also the same as Theorem 6.9, but we explain it again here nevertheless, for the completeness of this proof.

Let $F(\mathbf{x}) := \frac{1}{n} \sum_{i=1}^n f_i(\mathbf{x})$ be such that its minimizer is at the origin. This can be assumed without loss of generality because we can shift the coordinates appropriately, similar to (Safran and Shamir, 2020). Since the f_i are convex quadratics, we can write them as $f_i(\mathbf{x}) = \frac{1}{2} \mathbf{x}^\top \mathbf{A}_i \mathbf{x} - \mathbf{b}_i^\top \mathbf{x} + c_i$, where $\mathbf{A}_i$ are symmetric, positive-semidefinite matrices. We can omit the constants c_i because they do not affect the minimizer or the gradients. Because we assume that the minimizer of $F(\mathbf{x})$ is at the origin, we get that

$$\sum_{i=1}^n \mathbf{b}_i = \mathbf{0}. \quad (\text{C.18})$$

Let $\sigma^k = (\sigma_1^k, \sigma_2^k, \dots, \sigma_n^k)$ be the random permutation of $(1, 2, \dots, n)$ sampled in epoch

$2k - 1$. Then epoch $2k - 1$ sees the n functions in the reverse sequence:

$$\left(\frac{1}{2} \mathbf{x}^\top \mathbf{A}_{\sigma_1^k} \mathbf{x} - \mathbf{b}_{\sigma_1^k}^\top \mathbf{x}, \frac{1}{2} \mathbf{x}^\top \mathbf{A}_{\sigma_2^k} \mathbf{x} - \mathbf{b}_{\sigma_2^k}^\top \mathbf{x}, \dots, \frac{1}{2} \mathbf{x}^\top \mathbf{A}_{\sigma_n^k} \mathbf{x} - \mathbf{b}_{\sigma_n^k}^\top \mathbf{x} \right),$$

whereas epoch $2k$ sees the n functions in the reverse sequence:

$$\left(\frac{1}{2} \mathbf{x}^\top \mathbf{A}_{\sigma_n^k} \mathbf{x} - \mathbf{b}_{\sigma_n^k}^\top \mathbf{x}, \frac{1}{2} \mathbf{x}^\top \mathbf{A}_{\sigma_{n-1}^k} \mathbf{x} - \mathbf{b}_{\sigma_{n-1}^k}^\top \mathbf{x}, \dots, \frac{1}{2} \mathbf{x}^\top \mathbf{A}_{\sigma_1^k} \mathbf{x} - \mathbf{b}_{\sigma_1^k}^\top \mathbf{x} \right).$$

We define $\mathbf{S}_i^k := \alpha \mathbf{A}_{\sigma_i^k}$ and $\mathbf{t}_i^k = \alpha \mathbf{b}_{\sigma_i^k}$ for convenience of notation. We start off by computing the progress made during an even indexed epoch. Since the even epochs use the reverse permutation of σ^k , we get

$$\begin{aligned} \mathbf{x}_0^{2k+1} &= \mathbf{x}_n^{2k} \\ &= \mathbf{x}_{n-1}^{2k} - \alpha \left(\mathbf{A}_{\sigma_1^k} \mathbf{x}_{n-1}^{2k} - \mathbf{b}_{\sigma_1^k} \right) \quad (f_{\sigma_1^k} \text{ used at the last iteration of epoch } 2k.) \\ &= (\mathbf{I} - \alpha \mathbf{A}_{\sigma_1^k}) \mathbf{x}_{n-1}^{2k} + \alpha \mathbf{b}_{\sigma_1^k} \\ &= (\mathbf{I} - \mathbf{S}_1^k) \mathbf{x}_{n-1}^{2k} + \mathbf{t}_1^k. \end{aligned}$$

We recursively apply the same procedure as above to the whole epoch to get the following

$$\begin{aligned} \mathbf{x}_0^{2k+1} &= (\mathbf{I} - \mathbf{S}_1^k) \mathbf{x}_{n-1}^{2k} + \mathbf{t}_1^k \\ &= (\mathbf{I} - \mathbf{S}_1^k) ((\mathbf{I} - \mathbf{S}_2^k) \mathbf{x}_{n-2}^{2k} + \mathbf{t}_2^k) + \mathbf{t}_1^k \\ &= (\mathbf{I} - \mathbf{S}_1^k) (\mathbf{I} - \mathbf{S}_2^k) \mathbf{x}_{n-2}^{2k} + (\mathbf{I} - \mathbf{S}_1^k) \mathbf{t}_2^k + \mathbf{t}_1^k \\ &\quad \vdots \\ &= \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i^k) \right) \mathbf{x}_0^{2k} + \sum_{i=1}^n \left(\prod_{j=1}^{n-i} (\mathbf{I} - \mathbf{S}_j^k) \right) \mathbf{t}_{n+1-i}^k, \end{aligned} \tag{C.19}$$

where the product of matrices $\{\mathbf{M}_i\}$ is defined as $\prod_{i=l}^m \mathbf{M}_i = \mathbf{M}_l \mathbf{M}_{l+1} \dots \mathbf{M}_m$ if $m \geq l$ and 1 otherwise. Similar to Eq. (C.19), we can compute the progress made during an odd indexed epoch. Recall that the only difference will be that the odd indexed epochs see the permutations in the order $(\sigma_1^k, \sigma_2^k, \dots, \sigma_n^k)$ instead of $(\sigma_n^k, \sigma_{n-1}^k, \dots, \sigma_1^k)$. After doing the computation, we get the following equation:

$$\mathbf{x}_0^{2k} = \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}^k) \right) \mathbf{x}_0^{2k-1} + \sum_{i=1}^n \left(\prod_{j=1}^{n-i} (\mathbf{I} - \mathbf{S}_{n-j+1}^k) \right) \mathbf{t}_i^k.$$

Combining the results above, we can get the total progress made after the pair of epoch

$2k - 1$ and $2k$:

$$\begin{aligned}
\mathbf{x}_0^{2k+1} &= \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i^k) \right) \mathbf{x}_0^{2k} + \sum_{i=1}^n \left(\prod_{j=1}^{n-i} (\mathbf{I} - \mathbf{S}_j^k) \right) \mathbf{t}_{n+1-i}^k \\
&= \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i^k) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}^k) \right) \mathbf{x}_0^{2k-1} \\
&\quad + \left(\prod_{j=1}^n (\mathbf{I} - \mathbf{S}_j^k) \right) \sum_{i=1}^n \left(\prod_{j=1}^{n-i} (\mathbf{I} - \mathbf{S}_{n+1-j}^k) \right) \mathbf{t}_i^k + \sum_{i=1}^n \left(\prod_{j=1}^{n-i} (\mathbf{I} - \mathbf{S}_j^k) \right) \mathbf{t}_{n+1-i}^k. \quad (\text{C.20})
\end{aligned}$$

In the sum above, the first term will have an exponential decay, hence we need to control the next two terms. Similar to Theorem 6.9, we denote the sum of the terms as $\mathbf{z}^k$:

$$\begin{aligned}
\mathbf{z}^k &:= \left(\prod_{j=1}^n (\mathbf{I} - \mathbf{S}_j^k) \right) \sum_{i=1}^n \left(\prod_{j=1}^{n-i} (\mathbf{I} - \mathbf{S}_{n+1-j}^k) \right) \mathbf{t}_i^k + \sum_{i=1}^n \left(\prod_{j=1}^{n-i} (\mathbf{I} - \mathbf{S}_j^k) \right) \mathbf{t}_{n+1-i}^k \\
&= \sum_{i=1}^n \left(\prod_{j=1}^n (\mathbf{I} - \mathbf{S}_j^k) \right) \left(\prod_{j=1}^{n-i} (\mathbf{I} - \mathbf{S}_{n+1-j}^k) \right) \mathbf{t}_i^k + \sum_{i=1}^n \left(\prod_{j=1}^{n-i} (\mathbf{I} - \mathbf{S}_j^k) \right) \mathbf{t}_{n+1-i}^k.
\end{aligned}$$

To see where the iterates end up after K epochs, we simply set $2k = K$ in Eq. C.20 and then keep applying the equation recursively to preceding epochs. Then, we get

$$\begin{aligned}
\mathbf{x}_n^K &= \mathbf{x}_0^{K+1} = \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i^{\frac{K}{2}}) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}^{\frac{K}{2}}) \right) \mathbf{x}_0^{K-1} + \mathbf{z}^{\frac{K}{2}} \\
&= \left(\left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i^{\frac{K}{2}}) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}^{\frac{K}{2}}) \right) \right) \\
&\quad \times \left(\left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i^{\frac{K}{2}-1}) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}^{\frac{K}{2}-1}) \right) \right) \mathbf{x}_0^{K-3} \\
&\quad + \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i^{\frac{K}{2}}) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}^{\frac{K}{2}}) \right) \mathbf{z}^{\frac{K}{2}-1} + \mathbf{z}^{\frac{K}{2}} \\
&\quad \vdots \\
&= \left(\prod_{k=0}^{\frac{K}{2}-1} \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i^{\frac{K}{2}-k}) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}^{\frac{K}{2}-k}) \right) \right) \mathbf{x}_0^1 \\
&\quad + \sum_{k=0}^{\frac{K}{2}-1} \left(\prod_{l=0}^{k-1} \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i^{\frac{K}{2}-l}) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}^{\frac{K}{2}-l}) \right) \right) \mathbf{z}^{\frac{K}{2}-k}.
\end{aligned}$$

Taking squared norms and expectations on both sides, we get

$$\begin{aligned}
\mathbb{E}[\|\mathbf{x}_n^K\|^2] &= \mathbb{E} \left[\left\| \left(\prod_{k=0}^{\frac{K}{2}-1} \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i^{\frac{K}{2}-k}) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}^{\frac{K}{2}-k}) \right) \right) \mathbf{x}_0^1 \right. \right. \\
&\quad \left. \left. + \sum_{k=0}^{\frac{K}{2}-1} \left(\prod_{l=0}^{k-1} \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i^{\frac{K}{2}-l}) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}^{\frac{K}{2}-l}) \right) \right) \mathbf{z}^{\frac{K}{2}-k} \right\|^2 \right] \\
&\leq 2\mathbb{E} \left[\left\| \left(\prod_{k=0}^{\frac{K}{2}-1} \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i^{\frac{K}{2}-k}) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}^{\frac{K}{2}-k}) \right) \right) \mathbf{x}_0^1 \right\|^2 \right] \\
&\quad + 2\mathbb{E} \left[\left\| \sum_{k=0}^{\frac{K}{2}-1} \left(\prod_{l=0}^{k-1} \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i^{\frac{K}{2}-l}) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}^{\frac{K}{2}-l}) \right) \right) \mathbf{z}^{\frac{K}{2}-k} \right\|^2 \right], \quad (\text{C.21})
\end{aligned}$$

where we used the fact that $(a+b)^2 \leq 2a^2 + 2b^2$. Next, we expand the second term above to get

$$\begin{aligned}
\mathbb{E}[\|\mathbf{x}_n^K\|^2] &\leq 2\mathbb{E} \left[\left\| \left(\prod_{k=0}^{\frac{K}{2}-1} \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i^{\frac{K}{2}-k}) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}^{\frac{K}{2}-k}) \right) \right) \mathbf{x}_0^1 \right\|^2 \right] \\
&\quad + 2 \sum_{k=0}^{\frac{K}{2}-1} \mathbb{E} \left[\left\| \left(\prod_{l=0}^{k-1} \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i^{\frac{K}{2}-l}) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}^{\frac{K}{2}-l}) \right) \right) \mathbf{z}^{\frac{K}{2}-k} \right\|^2 \right] \\
&\quad + 4 \sum_{0 \leq k' < k \leq \frac{K}{2}-1} \mathbb{E} \left[\left\langle \left(\prod_{l=0}^{k-1} \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i^{\frac{K}{2}-l}) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}^{\frac{K}{2}-l}) \right) \right) \mathbf{z}^{\frac{K}{2}-k}, \right. \right. \\
&\quad \left. \left. \left(\prod_{l=0}^{k'-1} \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i^{\frac{K}{2}-l}) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}^{\frac{K}{2}-l}) \right) \right) \mathbf{z}^{\frac{K}{2}-k'} \right\rangle \right]. \quad (\text{C.22})
\end{aligned}$$

We handle each of the three terms separately. Let

$$\mathbf{N}_k := \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i^{\frac{K}{2}-k}) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}^{\frac{K}{2}-k}) \right).$$

The first term can be written as:

$$\mathbb{E} \left[\left\| \left(\prod_{k=0}^{\frac{K}{2}-1} \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i^{\frac{K}{2}-k}) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}^{\frac{K}{2}-k}) \right) \right) \mathbf{x}_0^1 \right\|^2 \right] = \mathbb{E} \left[\left\| \left(\prod_{k=0}^{\frac{K}{2}-1} \mathbf{N}_k \right) \mathbf{x}_0^1 \right\|^2 \right]$$

$$\begin{aligned}
&= \mathbb{E} \left[(\mathbf{x}_0^1)^\top \left(\prod_{k=0}^{\frac{K}{2}-1} \mathbf{N}_k \right)^\top \left(\prod_{k=0}^{\frac{K}{2}-1} \mathbf{N}_k \right) \mathbf{x}_0^1 \right] \\
&= \mathbb{E} \left[(\mathbf{x}_0^1)^\top \left(\prod_{k=1}^{\frac{K}{2}-1} \mathbf{N}_k \right)^\top \mathbf{N}_0^\top \mathbf{N}_0 \left(\prod_{k=1}^{\frac{K}{2}-1} \mathbf{N}_k \right) \mathbf{x}_0^1 \right] \\
&= \mathbb{E} \left[(\mathbf{x}_0^1)^\top \left(\prod_{k=1}^{\frac{K}{2}-1} \mathbf{N}_k \right)^\top \mathbb{E} [\mathbf{N}_0^\top \mathbf{N}_0] \left(\prod_{k=1}^{\frac{K}{2}-1} \mathbf{N}_k \right) \mathbf{x}_0^1 \right],
\end{aligned}$$

where, in the last line, we used the fact that the permutations in every epoch are independent of the permutations in other epochs. Next, we have the following lemma that bounds the spectral norm of $\mathbb{E} [\mathbf{N}_k^\top \mathbf{N}_k]$ for any $k \in [0, \frac{K}{2} - 1]$:

Lemma C.10. For any $0 \leq \alpha \leq \frac{3}{16nL} \min\{1, \sqrt{\frac{n}{\kappa}}\}$,

$$\|\mathbb{E}[\mathbf{N}_k^\top \mathbf{N}_k]\| \leq 1 - \alpha n \mu$$

Note that $K \geq 55\kappa \log(nK) \max\{1, \sqrt{\frac{n}{\kappa}}\} \implies \alpha \leq \frac{3}{16nL} \min\{1, \frac{n}{\kappa}\}$, and thus this lemma. Using it, we get

$$\begin{aligned}
&\mathbb{E} \left[\left\| \left(\prod_{k=0}^{\frac{K}{2}-1} \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i^{\frac{K}{2}-k}) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}^{\frac{K}{2}-k}) \right) \right) \mathbf{x}_0^1 \right\|^2 \right] \\
&= \mathbb{E} \left[(\mathbf{x}_0^1)^\top \left(\prod_{k=1}^{\frac{K}{2}-1} \mathbf{N}_k \right)^\top \mathbb{E} [\mathbf{N}_0^\top \mathbf{N}_0] \left(\prod_{k=1}^{\frac{K}{2}-1} \mathbf{N}_k \right) \mathbf{x}_0^1 \right] \\
&\leq (1 - n\alpha\mu) \mathbb{E} \left[(\mathbf{x}_0^1)^\top \left(\prod_{k=1}^{\frac{K}{2}-1} \mathbf{N}_k \right)^\top \left(\prod_{k=1}^{\frac{K}{2}-1} \mathbf{N}_k \right) \mathbf{x}_0^1 \right] \\
&= (1 - n\alpha\mu) \mathbb{E} \left[(\mathbf{x}_0^1)^\top \left(\prod_{k=2}^{\frac{K}{2}-1} \mathbf{N}_k \right)^\top \mathbb{E} [\mathbf{N}_1^\top \mathbf{N}_1] \left(\prod_{k=2}^{\frac{K}{2}-1} \mathbf{N}_k \right) \mathbf{x}_0^1 \right] \\
&\quad \vdots \\
&\leq (1 - n\alpha\mu)^{K/2} \|\mathbf{x}_0^1\|^2.
\end{aligned} \tag{C.23}$$

Next we look at the second term in Ineq. (C.22):

$$\begin{aligned}
& \mathbb{E} \left[\left\| \left(\prod_{l=0}^{k-1} \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i^{\frac{K}{2}-l}) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}^{\frac{K}{2}-l}) \right) \right) \mathbf{z}^{\frac{K}{2}-k} \right\|^2 \right] \\
& \leq \mathbb{E} \left[\left(\prod_{l=0}^{k-1} \left(\prod_{i=1}^n \|\mathbf{I} - \mathbf{S}_i^{\frac{K}{2}-l}\|^2 \right) \left(\prod_{i=1}^n \|\mathbf{I} - \mathbf{S}_{n-i+1}^{\frac{K}{2}-l}\|^2 \right) \right) \|\mathbf{z}^{\frac{K}{2}-k}\|^2 \right] \\
& \leq \mathbb{E} \left[\|\mathbf{z}^{\frac{K}{2}-k}\|^2 \right],
\end{aligned}$$

where in the last step we used that $\|\mathbf{I} - \mathbf{S}_i^k\| \leq 1$ for all i, k . To see why this is true, recall that $\mathbf{S}_i^k = \alpha \mathbf{A}_{\sigma_i^k}$. Further by Assumption 6.2, $\|\mathbf{A}_{\sigma_i^k}\| \leq L$ and hence as long as $\alpha \leq 1/L$, we have that $\|\mathbf{I} - \mathbf{S}_i^k\| \leq 1$.

Next, note that for any k we can apply Lemma C.6 on $\mathbb{E}[\|\mathbf{z}^{\frac{K}{2}-k}\|^2]$. Hence, we get the following bound on the second term:

$$\begin{aligned}
& 2 \sum_{k=0}^{\frac{K}{2}-1} \mathbb{E} \left[\left\| \left(\prod_{l=0}^{k-1} \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i^{\frac{K}{2}-l}) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}^{\frac{K}{2}-l}) \right) \right) \mathbf{z}^{\frac{K}{2}-k} \right\|^2 \right] \\
& \leq 2 \frac{K}{2} (2n^2 \alpha^4 L^2 (G^*)^2 + 170n^5 \alpha^6 L^4 (G^*)^2 \log n). \tag{C.24}
\end{aligned}$$

Finally, we focus on the third term in Ineq. (C.22). We have the following lemma that gives an upper bound for it:

Lemma C.11. *Let $\alpha \leq \frac{1}{2nL}$ and $n > 6$. Then for $0 \leq k' < k \leq \frac{K}{2} - 1$,*

$$\begin{aligned}
& \mathbb{E} \left[\left\langle \left(\prod_{l=0}^{k-1} \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i^{\frac{K}{2}-l}) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}^{\frac{K}{2}-l}) \right) \right) \mathbf{z}^{\frac{K}{2}-k}, \right. \right. \\
& \quad \left. \left(\prod_{l=0}^{k'-1} \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i^{\frac{K}{2}-l}) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}^{\frac{K}{2}-l}) \right) \right) \mathbf{z}^{\frac{K}{2}-k'} \right\rangle \right] \\
& \leq 1000n^2 \alpha^4 L^2 (G^*)^2 + 2000n^6 \alpha^7 L^5 (G^*)^2 \log n.
\end{aligned}$$

Using Lemma C.11, and inequalities (C.23) and (C.24) in Ineq. (C.22), we get

$$\begin{aligned}
\mathbb{E}[\|\mathbf{x}_n^K\|^2] & \leq 2(1 - n\alpha\mu)^{K/2} \|\mathbf{x}_0^1\|^2 + 2n^2 K \alpha^4 L^2 (G^*)^2 + 170n^5 K \alpha^6 L^4 (G^*)^2 \log n \\
& \quad + 1000n^2 K^2 \alpha^4 L^2 (G^*)^2 + 2000n^6 K^2 \alpha^7 L^5 (G^*)^2 \log n \\
& \leq 2(1 - n\alpha\mu)^{K/2} \|\mathbf{x}_0^1\|^2 + 1002n^2 K \alpha^4 L^2 (G^*)^2 + 2170n^6 K^2 \alpha^7 L^4 (G^*)^2 \log n.
\end{aligned}$$

Substituting $\alpha = \frac{10 \log n K}{\mu n K}$ gives us the desired result.

C.6.1 Proof of Lemma C.10

Recall that we have defined $\mathbf{N}_k := \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i^{\frac{K}{2}-k}) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}^{\frac{K}{2}-k}) \right)$. Hence,

$$\begin{aligned}
 \mathbf{N}_k^\top &= \left(\left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i^{\frac{K}{2}-k}) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}^{\frac{K}{2}-k}) \right) \right)^\top \\
 &= \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}^{\frac{K}{2}-k}) \right)^\top \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i^{\frac{K}{2}-k}) \right)^\top \\
 &= \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i^{\frac{K}{2}-k})^\top \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}^{\frac{K}{2}-k})^\top \right) \\
 &= \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i^{\frac{K}{2}-k}) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}^{\frac{K}{2}-k}) \right) \\
 &= \mathbf{N}_k,
 \end{aligned}$$

where we used the fact that $\mathbf{S}_i^{\frac{K}{2}-k}$ are symmetric. Hence $\mathbf{N}_k$ is symmetric. Then,

$$\begin{aligned}
 \|\mathbb{E}[\mathbf{N}_k^\top \mathbf{N}_k]\| &= \max_{\mathbf{x}: \|\mathbf{x}\|=1} \mathbb{E}[\mathbf{x}^\top \mathbf{N}_k^\top \mathbf{N}_k \mathbf{x}] \\
 &= \max_{\mathbf{x}: \|\mathbf{x}\|=1} \mathbb{E}[\mathbf{x}^\top \mathbf{N}_k \mathbf{N}_k \mathbf{x}].
 \end{aligned}$$

Next, note that $\|\mathbf{N}_k\| \leq 1$ as long as $\alpha \leq \frac{1}{nL}$. This combined with the fact that $\mathbf{N}_k$ is symmetric gives us that $\mathbf{x}^\top \mathbf{N}_k \mathbf{N}_k \mathbf{x} \leq \mathbf{x}^\top \mathbf{N}_k \mathbf{x}$. Hence, we get

$$\begin{aligned}
 \|\mathbb{E}[\mathbf{N}_k^\top \mathbf{N}_k]\| &\leq \max_{\mathbf{x}: \|\mathbf{x}\|=1} \mathbb{E}[\mathbf{x}^\top \mathbf{N}_k \mathbf{x}] \\
 &= \|\mathbb{E}[\mathbf{N}_k]\| \\
 &= \left\| \mathbb{E} \left[\left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i^{\frac{K}{2}-k}) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}^{\frac{K}{2}-k}) \right) \right] \right\| \\
 &= \left\| \mathbb{E} \left[\left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i^{\frac{K}{2}-k}) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i^{\frac{K}{2}-k}) \right)^\top \right] \right\|.
 \end{aligned}$$

To complete the proof, we apply the following lemma from Ahn et al. (2020):

Lemma C.12 (Lemma 6, Ahn et al. (2020)). *For any $0 \leq \alpha \leq \frac{3}{16nL} \min\{1, \frac{n}{\kappa}\}$ and $k \in [K]$,*

$$\left\| \mathbb{E} \left[\left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i^{\frac{K}{2}-k}) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i^{\frac{K}{2}-k}) \right)^\top \right] \right\| \leq 1 - \alpha n \mu.$$

Remark C.13. *To avoid confusion, we remark here that, the term ' $\mathbf{S}_k$ ' in the paper Ahn et al. (2020)*

is the same as the term ' $\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i^k)$ ' in our paper, and hence the original lemma statement in their paper looks different from what is written above.

C.6.2 Proof of Lemma C.11

We begin by decomposing the product into product of independent terms, similar to proof of Lemma 8 in Ahn et al. (2020). However, after that we diverge from their proof since we use FLIPFLOP specific analysis.

$$\begin{aligned} & \mathbb{E} \left[\left\langle \left(\prod_{l=0}^{k-1} \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i^{\frac{K}{2}-l}) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}^{\frac{K}{2}-l}) \right) \right) \mathbf{z}^{\frac{K}{2}-k}, \right. \right. \\ & \quad \left. \left(\prod_{l=0}^{k'-1} \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i^{\frac{K}{2}-l}) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}^{\frac{K}{2}-l}) \right) \right) \mathbf{z}^{\frac{K}{2}-k'} \right\rangle \right] \\ &= \mathbb{E} \left[\left(\mathbf{z}^{\frac{K}{2}-k} \right)^\top \left(\prod_{l=k'}^{k+1} \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i^{\frac{K}{2}-l}) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}^{\frac{K}{2}-l}) \right) \right)^\top \right. \\ & \quad \left(\prod_{l=0}^{k'} \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i^{\frac{K}{2}-l}) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}^{\frac{K}{2}-l}) \right) \right)^\top \\ & \quad \left. \left(\prod_{l=0}^{k'-1} \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i^{\frac{K}{2}-l}) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}^{\frac{K}{2}-l}) \right) \right) \mathbf{z}^{\frac{K}{2}-k'} \right]. \end{aligned}$$

Since $k' < k$, we get that

$$\begin{aligned} & \left(\mathbf{z}^{\frac{K}{2}-k} \right)^\top ; \quad \left(\prod_{l=k'}^{k+1} \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i^{\frac{K}{2}-l}) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}^{\frac{K}{2}-l}) \right) \right)^\top ; \text{ and} \\ & \left(\prod_{l=0}^{k'} \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i^{\frac{K}{2}-l}) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}^{\frac{K}{2}-l}) \right) \right)^\top \\ & \quad \times \left(\prod_{l=0}^{k'-1} \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i^{\frac{K}{2}-l}) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}^{\frac{K}{2}-l}) \right) \right) \mathbf{z}^{\frac{K}{2}-k'} \end{aligned}$$

are independent. Hence, we can write the expectation as product of expectations:

$$\begin{aligned} & \mathbb{E} \left[\left\langle \left(\prod_{l=0}^{k-1} \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i^{\frac{K}{2}-l}) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}^{\frac{K}{2}-l}) \right) \right) \mathbf{z}^{\frac{K}{2}-k}, \right. \right. \\ & \quad \left. \left(\prod_{l=0}^{k'-1} \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i^{\frac{K}{2}-l}) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}^{\frac{K}{2}-l}) \right) \right) \mathbf{z}^{\frac{K}{2}-k'} \right\rangle \right] \end{aligned}$$

$$\begin{aligned}
&= \mathbb{E} \left[\left(\mathbf{z}^{\frac{K}{2}-k} \right)^\top \right] \mathbb{E} \left[\left(\prod_{l=k'}^{k+1} \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i^{\frac{K}{2}-l}) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}^{\frac{K}{2}-l}) \right) \right)^\top \right] \\
&\quad \mathbb{E} \left[\left(\prod_{l=0}^{k'} \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i^{\frac{K}{2}-l}) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}^{\frac{K}{2}-l}) \right) \right)^\top \right. \\
&\quad \left. \left(\prod_{l=0}^{k'-1} \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i^{\frac{K}{2}-l}) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}^{\frac{K}{2}-l}) \right) \right) \mathbf{z}^{\frac{K}{2}-k'} \right].
\end{aligned}$$

Applying the Cauchy-Schwarz inequality on the decomposition above, we get

$$\begin{aligned}
&\mathbb{E} \left[\left\langle \left(\prod_{l=0}^{k-1} \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i^{\frac{K}{2}-l}) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}^{\frac{K}{2}-l}) \right) \right) \mathbf{z}^{\frac{K}{2}-k}, \right. \right. \\
&\quad \left. \left. \left(\prod_{l=0}^{k'-1} \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i^{\frac{K}{2}-l}) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}^{\frac{K}{2}-l}) \right) \right) \mathbf{z}^{\frac{K}{2}-k'} \right\rangle \right] \\
&\leq \left\| \mathbb{E} \left[\mathbf{z}^{\frac{K}{2}-k} \right] \right\| \left\| \mathbb{E} \left[\prod_{l=k'}^{k+1} \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i^{\frac{K}{2}-l}) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}^{\frac{K}{2}-l}) \right) \right] \right\| \\
&\quad \left\| \mathbb{E} \left[\left(\prod_{l=0}^{k'} \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i^{\frac{K}{2}-l}) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}^{\frac{K}{2}-l}) \right) \right)^\top \right. \right. \\
&\quad \left. \left. \left(\prod_{l=0}^{k'-1} \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i^{\frac{K}{2}-l}) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}^{\frac{K}{2}-l}) \right) \right) \mathbf{z}^{\frac{K}{2}-k'} \right] \right\| \\
&\leq \left\| \mathbb{E} \left[\mathbf{z}^{\frac{K}{2}-k} \right] \right\| \mathbb{E} \left[\prod_{l=k'}^{k+1} \left(\prod_{i=1}^n \|\mathbf{I} - \mathbf{S}_i^{\frac{K}{2}-l}\| \right) \left(\prod_{i=1}^n \|\mathbf{I} - \mathbf{S}_{n-i+1}^{\frac{K}{2}-l}\| \right) \right] \\
&\quad \left\| \mathbb{E} \left[\left(\prod_{l=0}^{k'} \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i^{\frac{K}{2}-l}) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}^{\frac{K}{2}-l}) \right) \right)^\top \right. \right. \\
&\quad \left. \left. \left(\prod_{l=0}^{k'-1} \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i^{\frac{K}{2}-l}) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}^{\frac{K}{2}-l}) \right) \right) \mathbf{z}^{\frac{K}{2}-k'} \right] \right\| \\
&\leq \left\| \mathbb{E} \left[\mathbf{z}^{\frac{K}{2}-k} \right] \right\| \\
&\quad \left\| \mathbb{E} \left[\left(\prod_{l=0}^{k'} \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i^{\frac{K}{2}-l}) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}^{\frac{K}{2}-l}) \right) \right)^\top \right. \right. \\
&\quad \left. \left. \left(\prod_{l=0}^{k'-1} \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i^{\frac{K}{2}-l}) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}^{\frac{K}{2}-l}) \right) \right) \mathbf{z}^{\frac{K}{2}-k'} \right] \right\|,
\end{aligned}$$

where in the last step we used that $\|\mathbf{I} - \mathbf{S}_i^k\| \leq 1$ for all i, k . To see why this is true, recall that $\mathbf{S}_i^k = \alpha \mathbf{A}_{\sigma_i^k}$. Further by Assumption 6.2, $\|\mathbf{A}_{\sigma_i^k}\| \leq L$ and hence as long as $\alpha \leq 1/L$, we

have $\|\mathbf{I} - \mathbf{S}_i^k\| \leq 1$.

For the two terms in the product above, we have the following lemma:

Lemma C.14. *If $n > 6$ and $\alpha \leq \frac{1}{nL}$, then*

$$\|\mathbb{E}[\mathbf{z}^{\frac{K}{2}-k}]\| \leq 28n\alpha^2 LG^* + 9\alpha^5 L^4 n^4 G^* \sqrt{2n \log n}.$$

Lemma C.15. *If $n > 6$ and $\alpha \leq \frac{1}{2nL}$, then*

$$\begin{aligned} & \left\| \mathbb{E} \left[\left(\prod_{l=0}^{k'} \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i^{\frac{K}{2}-l}) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}^{\frac{K}{2}-l}) \right) \right)^{\top} \right. \right. \\ & \quad \left. \left(\prod_{l=0}^{k'-1} \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i^{\frac{K}{2}-l}) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}^{\frac{K}{2}-l}) \right) \right) \mathbf{z}^{\frac{K}{2}-k'} \right] \right\| \\ & \leq 32n\alpha^2 LG^* + 24\alpha^5 L^4 n^4 G^* \sqrt{2n \log n}. \end{aligned}$$

Finally, using these lemma we get

$$\begin{aligned} & \mathbb{E} \left[\left\langle \left(\prod_{l=0}^{k-1} \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i^{\frac{K}{2}-l}) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}^{\frac{K}{2}-l}) \right) \right) \mathbf{z}^{\frac{K}{2}-k}, \right. \right. \\ & \quad \left. \left(\prod_{l=0}^{k'-1} \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i^{\frac{K}{2}-l}) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}^{\frac{K}{2}-l}) \right) \right) \mathbf{z}^{\frac{K}{2}-k'} \right\rangle \right] \\ & \leq \left(28n\alpha^2 LG^* + 9\alpha^5 L^4 n^4 G^* \sqrt{2n \log n} \right) \left(32n\alpha^2 LG^* + 24\alpha^5 L^4 n^4 G^* \sqrt{2n \log n} \right) \\ & \leq 896n^2 \alpha^4 L^2 (G^*)^2 + 960\alpha^7 L^5 n^5 (G^*)^2 \sqrt{2n \log n} + 432\alpha^{10} L^8 n^9 (G^*)^2 \log n \\ & \leq 1000n^2 \alpha^4 L^2 (G^*)^2 + 2000\alpha^7 L^5 n^6 (G^*)^2 \log n. \end{aligned}$$

C.6.3 Proof of Lemma C.14

Since we are dealing with just a single epoch, we will skip the superscript. Using Lemma C.7, we get

$$\begin{aligned} \|\mathbb{E}[\mathbf{z}]\| &= \left\| \mathbb{E} \left[\sum_{i=1}^n \mathbf{S}_i \mathbf{t}_i \right] + \mathbb{E} \left[\sum_{j=n+1}^{2n-1} \sum_{l=1}^{j-1} \left(\prod_{p=1}^{l-1} (\mathbf{I} - \tilde{\mathbf{S}}_p) \right) \tilde{\mathbf{S}}_l \tilde{\mathbf{S}}_j \left(\sum_{i=1}^{2n-j} \mathbf{t}_i \right) \right] \right. \\ & \quad \left. + \mathbb{E} \left[\sum_{j=1}^{n-1} \sum_{l=1}^{j-1} \left(\prod_{p=1}^{l-1} (\mathbf{I} - \mathbf{S}_p) \right) \mathbf{S}_l \mathbf{S}_j \left(\sum_{i=1}^{n-j} \mathbf{t}_{n+1-i} \right) \right] \right\| \\ & \leq \sum_{i=1}^n \mathbb{E} [\|\mathbf{S}_i\| \|\mathbf{t}_i\|] + \sum_{j=n+1}^{2n-1} \sum_{l=1}^{j-1} \left\| \mathbb{E} \left[\left(\prod_{p=1}^{l-1} (\mathbf{I} - \tilde{\mathbf{S}}_p) \right) \tilde{\mathbf{S}}_l \tilde{\mathbf{S}}_j \left(\sum_{i=1}^{2n-j} \mathbf{t}_i \right) \right] \right\| \end{aligned}$$

$$+ \sum_{j=1}^{n-1} \sum_{l=1}^{j-1} \left\| \mathbb{E} \left[\left(\prod_{p=1}^{l-1} (\mathbf{I} - \mathbf{S}_p) \right) \mathbf{S}_l \mathbf{S}_j \left(\sum_{i=1}^{n-j} \mathbf{t}_{n+1-i} \right) \right] \right\|. \quad (\text{C.25})$$

Define $G^* := \max_i \|\mathbf{b}_i\|$. Then $\|\mathbf{S}_i\| \|\mathbf{t}_i\| = \|\alpha \mathbf{A}_{\sigma_i^k}\| \|\alpha \mathbf{b}_{\sigma_i^k}\| \leq \alpha^2 L G^*$ and hence

$$\sum_{i=1}^n \mathbb{E} [\|\mathbf{S}_i\| \|\mathbf{t}_i\|] \leq n \alpha^2 L G^*. \quad (\text{C.26})$$

Next we bound the other two terms. Using Eq. (C.16), we get that for any $l < j$,

$$\begin{aligned} & \mathbb{E} \left[\left(\prod_{p=1}^{l-1} (\mathbf{I} - \mathbf{S}_p) \right) \mathbf{S}_l \mathbf{S}_j \left(\sum_{i=1}^{n-j} \mathbf{t}_{n+1-i} \right) \right] \\ &= \mathbb{E} \left[\left(\mathbf{I} - \sum_{p=1}^{l-1} \mathbf{S}_p + \sum_{p=1}^{l-1} \sum_{q=1}^{p-1} \left(\prod_{r=1}^{q-1} (\mathbf{I} - \mathbf{S}_q) \right) \mathbf{S}_r \mathbf{S}_p \right) \mathbf{S}_l \mathbf{S}_j \left(\sum_{i=1}^{n-j} \mathbf{t}_{n+1-i} \right) \right] \\ &= \sum_{i=1}^{n-j} \mathbb{E} [\mathbf{S}_l \mathbf{S}_j \mathbf{t}_{n+1-i}] - \mathbb{E} \left[\sum_{p=1}^{l-1} \mathbf{S}_p \mathbf{S}_l \mathbf{S}_j \left(\sum_{i=1}^{n-j} \mathbf{t}_{n+1-i} \right) \right] \\ &\quad + \mathbb{E} \left[\sum_{p=1}^{l-1} \sum_{q=1}^{p-1} \left(\prod_{r=1}^{q-1} (\mathbf{I} - \mathbf{S}_q) \right) \mathbf{S}_r \mathbf{S}_p \mathbf{S}_l \mathbf{S}_j \left(\sum_{i=1}^{n-j} \mathbf{t}_{n+1-i} \right) \right] \\ &= \sum_{i>j,l} \mathbb{E} [\mathbf{S}_l \mathbf{S}_j \mathbf{t}_i] - \sum_{p<l,j<i} \mathbb{E} [\mathbf{S}_p \mathbf{S}_l \mathbf{S}_j \mathbf{t}_i] \\ &\quad + \sum_{p=1}^{l-1} \sum_{q=1}^{p-1} \mathbb{E} \left[\left(\prod_{r=1}^{q-1} (\mathbf{I} - \mathbf{S}_q) \right) \mathbf{S}_r \mathbf{S}_p \mathbf{S}_l \mathbf{S}_j \left(\sum_{i=1}^{n-j} \mathbf{t}_{n+1-i} \right) \right] \\ &= \sum_{i>j,l} \mathbb{E} [\mathbf{S}_l \mathbf{S}_j \mathbb{E}[\mathbf{t}_i | \mathbf{S}_l, \mathbf{S}_j]] - \sum_{p<l,j<i} \mathbb{E} [\mathbf{S}_p \mathbf{S}_l \mathbf{S}_j \mathbb{E}[\mathbf{t}_i | \mathbf{S}_l, \mathbf{S}_j, \mathbf{S}_p]] \\ &\quad + \sum_{p=1}^{l-1} \sum_{q=1}^{p-1} \mathbb{E} \left[\left(\prod_{r=1}^{q-1} (\mathbf{I} - \mathbf{S}_q) \right) \mathbf{S}_r \mathbf{S}_p \mathbf{S}_l \mathbf{S}_j \left(\sum_{i=1}^{n-j} \mathbf{t}_{n+1-i} \right) \right]. \end{aligned}$$

Since $\sum_{i=1}^n t_i = 0$, and we use uniform random permutations,

$$\mathbb{E}[\mathbf{t}_i | \mathbf{S}_l, \mathbf{S}_j] = \sum_{\substack{\mathbf{t}_i \neq \mathbf{t}_l, \\ \mathbf{t}_i \neq \mathbf{t}_j}} \frac{\mathbf{t}_i}{n-2} = \frac{-\mathbf{t}_j - \mathbf{t}_l}{n-2}.$$

Similarly, $\mathbb{E}[\mathbf{t}_i | \mathbf{S}_l, \mathbf{S}_j, \mathbf{S}_p] = \frac{-\mathbf{t}_j - \mathbf{t}_l - \mathbf{t}_p}{n-3}$. Hence,

$$\begin{aligned} & \left\| \mathbb{E} \left[\left(\prod_{p=1}^{l-1} (\mathbf{I} - \mathbf{S}_p) \right) \mathbf{S}_l \mathbf{S}_j \left(\sum_{i=1}^{n-j} \mathbf{t}_{n+1-i} \right) \right] \right\| \\ & \leq \sum_{i>j,l} \|\mathbb{E} [\mathbf{S}_l \mathbf{S}_j \mathbb{E}[\mathbf{t}_i | \mathbf{S}_l, \mathbf{S}_j]]\| + \sum_{p<l,j<i} \|\mathbb{E} [\mathbf{S}_p \mathbf{S}_l \mathbf{S}_j \mathbb{E}[\mathbf{t}_i | \mathbf{S}_l, \mathbf{S}_j, \mathbf{S}_p]]\| \end{aligned}$$

$$\begin{aligned}
& + \sum_{p=1}^{l-1} \sum_{q=1}^{p-1} \mathbb{E} \left[\left(\prod_{r=1}^{q-1} \|I - S_q\| \right) \|S_r\| \|S_p\| \|S_l\| \|S_j\| \left\| \sum_{i=1}^{n-j} t_{n+1-i} \right\| \right] \\
& \leq \sum_{i>j,l} \mathbb{E} \left[\|S_l\| \|S_j\| \frac{\|t_l + t_j\|}{n-2} \right] + \sum_{p<l,j<i} \mathbb{E} \left[\|S_p\| \|S_l\| \|S_j\| \frac{\|t_l + t_j + t_p\|}{n-3} \right] \\
& + \sum_{p=1}^{l-1} \sum_{q=1}^{p-1} \mathbb{E} \left[\left(\prod_{r=1}^{q-1} \|I - S_q\| \right) \|S_r\| \|S_p\| \|S_l\| \|S_j\| \left\| \sum_{i=1}^{n-j} t_{n+1-i} \right\| \right] \\
& \leq \frac{2n}{n-2} \alpha^3 L^2 G^* + \frac{3n^2}{2(n-3)} \alpha^4 L^3 G^* + \alpha^4 L^4 \sum_{p=1}^{l-1} \sum_{q=1}^{p-1} \mathbb{E} \left[\left\| \sum_{i=1}^{n-j} t_{n+1-i} \right\| \right] \\
& \quad \text{(Since } \|S_i\| \leq \alpha L, \|t_i\| \leq \alpha G^*) \\
& \leq 4\alpha^3 L^2 G^* + 3n\alpha^4 L^3 G^* + \alpha^5 L^4 n^2 G^* \sqrt{18n \log n}, \tag{C.27}
\end{aligned}$$

where we used Lemma C.8 and the assumption that $n > 6$ in the last step.

The third term in Ineq. (C.25) can be handled similarly. For any l, j :

$$\begin{aligned}
& \mathbb{E} \left[\left(\prod_{p=1}^{l-1} (I - \tilde{S}_p) \right) \tilde{S}_l \tilde{S}_j \left(\sum_{i=1}^{2n-j} t_i \right) \right] \\
& = \mathbb{E} \left[\left(I - \sum_{p=1}^{l-1} \tilde{S}_p + \sum_{p=1}^{l-1} \sum_{q=1}^{p-1} \left(\prod_{r=1}^{q-1} (I - \tilde{S}_q) \right) \tilde{S}_r \tilde{S}_p \right) \tilde{S}_l \tilde{S}_j \left(\sum_{i=1}^{2n-j} t_i \right) \right] \\
& = \sum_{i=1}^{2n-j} \mathbb{E} [\tilde{S}_l \tilde{S}_j t_i] - \sum_{p=1}^{l-1} \sum_{i=1}^{2n-j} \mathbb{E} [\tilde{S}_p \tilde{S}_l \tilde{S}_j t_i] \\
& \quad + \mathbb{E} \left[\sum_{p=1}^{l-1} \sum_{q=1}^{p-1} \left(\prod_{r=1}^{q-1} (I - \tilde{S}_q) \right) \tilde{S}_r \tilde{S}_p \tilde{S}_l \tilde{S}_j \left(\sum_{i=1}^{2n-j} t_i \right) \right].
\end{aligned}$$

Now, it is easy to see that $i \neq j$ and $i \neq 2n - j + 1$. Then, if $i = l$ or $i = 2n - l + 1$ we use for that case the fact that $\|\mathbb{E}[\tilde{S}_l \tilde{S}_j t_i]\| \leq \alpha^3 L^2 G^*$. For all other i , we can again use that $\mathbb{E}[t_i | \tilde{S}_l, \tilde{S}_j] = \frac{-t_l - t_j}{n-2}$ if $l \leq n$ or $\mathbb{E}[t_i | \tilde{S}_l, \tilde{S}_j] = \frac{-t_{2n-l+1} - t_j}{n-2}$ otherwise. Similarly, we can bound $\|\mathbb{E}[\tilde{S}_p \tilde{S}_l \tilde{S}_j t_i]\|$. Proceeding in a way similar to how we derived Ineq. (C.27), we get

$$\begin{aligned}
& \left\| \mathbb{E} \left[\left(\prod_{p=1}^{l-1} (I - \tilde{S}_p) \right) \tilde{S}_l \tilde{S}_j \left(\sum_{i=1}^{2n-j} t_i \right) \right] \right\| \\
& \leq \sum_{i=1}^{2n-j} \|\mathbb{E}[\tilde{S}_l \tilde{S}_j t_i]\| + \sum_{p=1}^{l-1} \sum_{i=1}^{2n-j} \|\mathbb{E}[\tilde{S}_p \tilde{S}_l \tilde{S}_j t_i]\| \\
& \quad + \sum_{p=1}^{l-1} \sum_{q=1}^{p-1} \left\| \mathbb{E} \left[\left(\prod_{r=1}^{q-1} (I - \tilde{S}_q) \right) \tilde{S}_r \tilde{S}_p \tilde{S}_l \tilde{S}_j \left(\sum_{i=1}^{2n-j} t_i \right) \right] \right\|
\end{aligned}$$

$$\begin{aligned}
&\leq \alpha^3 L^2 G^* + \frac{2n}{n-2} \alpha^3 L^2 G^* + \alpha^4 L^3 G^* + n \alpha^4 L^3 G^* + \frac{3n^2}{2(n-3)} \alpha^4 L^3 G^* \\
&\quad + \sum_{p=1}^{l-1} \sum_{q=1}^{p-1} \left\| \mathbb{E} \left[\left(\prod_{r=1}^{q-1} (\mathbf{I} - \tilde{\mathbf{S}}_q) \right) \tilde{\mathbf{S}}_r \tilde{\mathbf{S}}_p \tilde{\mathbf{S}}_l \tilde{\mathbf{S}}_j \left(\sum_{i=1}^{2n-j} \mathbf{t}_i \right) \right] \right\| \\
&\leq 5\alpha^3 L^2 G^* + 5n\alpha^4 L^3 G^* \\
&\quad + \sum_{p=1}^{l-1} \sum_{q=1}^{p-1} \left\| \mathbb{E} \left[\left(\prod_{r=1}^{q-1} \|\mathbf{I} - \tilde{\mathbf{S}}_q\| \right) \|\tilde{\mathbf{S}}_r\| \|\tilde{\mathbf{S}}_p\| \|\tilde{\mathbf{S}}_l\| \|\tilde{\mathbf{S}}_j\| \left(\sum_{i=1}^{2n-j} \mathbf{t}_i \right) \right] \right\| \\
&\leq 5\alpha^3 L^2 G^* + 5n\alpha^4 L^3 G^* + \alpha^5 L^4 n^2 G^* \sqrt{18n \log n}. \tag{C.28}
\end{aligned}$$

Substituting Ineq. (C.26), (C.27) and (C.28) into (C.25), we get

$$\begin{aligned}
\|\mathbb{E}[\mathbf{z}]\| &\leq n\alpha^2 L G^* + 10n^2 \alpha^3 L^2 G^* + 10n^3 \alpha^4 L^3 G^* + 2\alpha^5 L^4 n^4 G^* \sqrt{18n \log n} \\
&\quad + 4n^2 \alpha^3 L^2 G^* + 3n^3 \alpha^4 L^3 G^* + \alpha^5 L^4 n^4 G^* \sqrt{18n \log n} \\
&\leq 28n\alpha^2 L G^* + 9\alpha^5 L^4 n^4 G^* \sqrt{2n \log n},
\end{aligned}$$

where we used the assumption that $\alpha \leq \frac{1}{nL}$ in the last step.

C.6.4 Proof of Lemma C.15

Define the matrix $\mathbf{M}$ as

$$\begin{aligned}
\mathbf{M} &:= \left(\prod_{l=0}^{k'-1} \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i^{\frac{K}{2}-l}) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}^{\frac{K}{2}-l}) \right) \right)^\top \\
&\quad \times \left(\prod_{l=0}^{k'-1} \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i^{\frac{K}{2}-l}) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}^{\frac{K}{2}-l}) \right) \right).
\end{aligned}$$

Since $\mathbf{M}$ is independent of $\left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i^{\frac{K}{2}-k'}) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}^{\frac{K}{2}-k'}) \right)$ and $\mathbf{z}^{\frac{K}{2}-k'}$, using the tower rule, we get

$$\begin{aligned}
&\mathbb{E} \left[\left(\prod_{l=0}^{k'-1} \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i^{\frac{K}{2}-l}) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}^{\frac{K}{2}-l}) \right) \right)^\top \right. \\
&\quad \left. \left(\prod_{l=0}^{k'-1} \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i^{\frac{K}{2}-l}) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}^{\frac{K}{2}-l}) \right) \right) \mathbf{z}^{\frac{K}{2}-k'} \right] \\
&= \mathbb{E} \left[\left(\left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i^{\frac{K}{2}-k'}) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}^{\frac{K}{2}-k'}) \right) \right)^\top \mathbb{E}[\mathbf{M}] \mathbf{z}^{\frac{K}{2}-k'} \right].
\end{aligned}$$

We will now drop the superscript $\frac{K}{2} - k'$ for convenience. Hence, we need to control the following term:

$$\mathbb{E} \left[\left(\left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}) \right) \right)^\top \mathbb{E}[\mathbf{M}] \mathbf{z} \right]$$

We define $(\tilde{\mathbf{S}}_1, \dots, \tilde{\mathbf{S}}_{2n})$ as $(\tilde{\mathbf{S}}_1, \dots, \tilde{\mathbf{S}}_n) = (\mathbf{S}_1, \dots, \mathbf{S}_n)$ and $(\tilde{\mathbf{S}}_{n+1}, \dots, \tilde{\mathbf{S}}_{2n}) = (\mathbf{S}_n, \dots, \mathbf{S}_1)$. Then, we use Eq. (C.14) to get

$$\begin{aligned} \left(\left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}) \right) \right) &= \prod_{i=1}^{2n} (\mathbf{I} - \tilde{\mathbf{S}}_i) \\ &= \mathbf{I} - \sum_{j=1}^{2n} \tilde{\mathbf{S}}_j + \sum_{j=1}^{2n} \sum_{l=1}^{j-1} \left(\prod_{p=1}^{l-1} (\mathbf{I} - \tilde{\mathbf{S}}_p) \right) \tilde{\mathbf{S}}_l \tilde{\mathbf{S}}_j. \end{aligned}$$

Note that $\mathbf{I} - \sum_{j=1}^{2n} \tilde{\mathbf{S}}_j$ is a constant matrix. Since $\tilde{\mathbf{S}}_j = \alpha \mathbf{A}_{\sigma_j}$, we have that $\|\tilde{\mathbf{S}}_j\| \leq \alpha L$ by Assumption 6.2. Hence, $\alpha \leq \frac{1}{2nL}$ then $\|\mathbf{I} - \sum_{j=1}^{2n} \tilde{\mathbf{S}}_j\| \leq 1$. Further, $\alpha \leq 1/L$ implies that $\|\mathbf{I} - \mathbf{S}_i\| \leq 1$, which implies $\|\mathbf{M}\| \leq 1$. Hence,

$$\begin{aligned} &\left\| \mathbb{E} \left[\left(\left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}) \right) \right)^\top \mathbb{E}[\mathbf{M}] \mathbf{z} \right] \right\| \\ &\leq \left\| \mathbb{E} \left[\left(\mathbf{I} - \sum_{j=1}^{2n} \tilde{\mathbf{S}}_j \right)^\top \mathbb{E}[\mathbf{M}] \mathbf{z} \right] \right\| \\ &\quad + \left\| \mathbb{E} \left[\left(\sum_{j=1}^{2n} \sum_{l=1}^{j-1} \left(\prod_{p=1}^{l-1} (\mathbf{I} - \tilde{\mathbf{S}}_p) \right) \tilde{\mathbf{S}}_l \tilde{\mathbf{S}}_j \right)^\top \mathbb{E}[\mathbf{M}] \mathbf{z} \right] \right\| \\ &= \left\| \left(\mathbf{I} - \sum_{j=1}^{2n} \tilde{\mathbf{S}}_j \right)^\top \mathbb{E}[\mathbf{M}] \mathbb{E}[\mathbf{z}] \right\| + \left\| \mathbb{E} \left[\left(\sum_{j=1}^{2n} \sum_{l=1}^{j-1} \left(\prod_{p=1}^{l-1} (\mathbf{I} - \tilde{\mathbf{S}}_p) \right) \tilde{\mathbf{S}}_l \tilde{\mathbf{S}}_j \right)^\top \mathbb{E}[\mathbf{M}] \mathbf{z} \right] \right\| \\ &\leq \|\mathbb{E}[\mathbf{z}]\| + \left\| \mathbb{E} \left[\left(\sum_{j=1}^{2n} \sum_{l=1}^{j-1} \left(\prod_{p=1}^{l-1} (\mathbf{I} - \tilde{\mathbf{S}}_p) \right) \tilde{\mathbf{S}}_l \tilde{\mathbf{S}}_j \right)^\top \mathbb{E}[\mathbf{M}] \mathbf{z} \right] \right\|. \end{aligned}$$

We can apply Lemma C.14 to bound $\|\mathbb{E}[\mathbf{z}]\|$. So, we focus on the other term. Using Lemma C.7,

$$\left\| \mathbb{E} \left[\left(\sum_{j=1}^{2n} \sum_{l=1}^{j-1} \left(\prod_{p=1}^{l-1} (\mathbf{I} - \tilde{\mathbf{S}}_p) \right) \tilde{\mathbf{S}}_l \tilde{\mathbf{S}}_j \right)^\top \mathbb{E}[\mathbf{M}] \mathbf{z} \right] \right\|$$

$$\begin{aligned}
&\leq \left\| \mathbb{E} \left[\left(\sum_{j=1}^{2n} \sum_{l=1}^{j-1} \left(\prod_{p=1}^{l-1} (\mathbf{I} - \tilde{\mathbf{S}}_p) \right) \tilde{\mathbf{S}}_l \tilde{\mathbf{S}}_j \right)^\top \mathbb{E}[\mathbf{M}] \left(\sum_{i=1}^n \mathbf{S}_i \mathbf{t}_i \right) \right] \right\| \\
&\quad + \left\| \mathbb{E} \left[\left(\sum_{j=1}^{2n} \sum_{l=1}^{j-1} \left(\prod_{p=1}^{l-1} (\mathbf{I} - \tilde{\mathbf{S}}_p) \right) \tilde{\mathbf{S}}_l \tilde{\mathbf{S}}_j \right)^\top \mathbb{E}[\mathbf{M}] \right. \right. \\
&\quad \quad \left. \left. \times \left(\sum_{j=n+1}^{2n-1} \sum_{l=1}^{j-1} \left(\prod_{p=1}^{l-1} (\mathbf{I} - \tilde{\mathbf{S}}_p) \right) \tilde{\mathbf{S}}_l \tilde{\mathbf{S}}_j \left(\sum_{i=1}^{2n-j} \mathbf{t}_i \right) \right) \right] \right\| \\
&\quad + \left\| \mathbb{E} \left[\left(\sum_{j=1}^{2n} \sum_{l=1}^{j-1} \left(\prod_{p=1}^{l-1} (\mathbf{I} - \tilde{\mathbf{S}}_p) \right) \tilde{\mathbf{S}}_l \tilde{\mathbf{S}}_j \right)^\top \mathbb{E}[\mathbf{M}] \right. \right. \\
&\quad \quad \left. \left. \times \left(\sum_{j=1}^{n-1} \sum_{l=1}^{j-1} \left(\prod_{p=1}^{l-1} (\mathbf{I} - \mathbf{S}_p) \right) \mathbf{S}_l \mathbf{S}_j \left(\sum_{i=1}^{n-j} \mathbf{t}_{n+1-i} \right) \right) \right] \right\| \\
&= \left\| \mathbb{E} \left[\left(\sum_{j=1}^{2n} \sum_{l=1}^{j-1} \left(\prod_{p=1}^{l-1} (\mathbf{I} - \tilde{\mathbf{S}}_p) \right) \tilde{\mathbf{S}}_l \tilde{\mathbf{S}}_j \right)^\top \right] \mathbb{E}[\mathbf{M}] \left(\sum_{i=1}^n \mathbf{S}_i \mathbf{t}_i \right) \right\| \\
&\quad + \left\| \mathbb{E} \left[\left(\sum_{j=1}^{2n} \sum_{l=1}^{j-1} \left(\prod_{p=1}^{l-1} (\mathbf{I} - \tilde{\mathbf{S}}_p) \right) \tilde{\mathbf{S}}_l \tilde{\mathbf{S}}_j \right)^\top \mathbb{E}[\mathbf{M}] \right. \right. \\
&\quad \quad \left. \left. \times \left(\sum_{j=n+1}^{2n-1} \sum_{l=1}^{j-1} \left(\prod_{p=1}^{l-1} (\mathbf{I} - \tilde{\mathbf{S}}_p) \right) \tilde{\mathbf{S}}_l \tilde{\mathbf{S}}_j \left(\sum_{i=1}^{2n-j} \mathbf{t}_i \right) \right) \right] \right\| \\
&\quad + \left\| \mathbb{E} \left[\left(\sum_{j=1}^{2n} \sum_{l=1}^{j-1} \left(\prod_{p=1}^{l-1} (\mathbf{I} - \tilde{\mathbf{S}}_p) \right) \tilde{\mathbf{S}}_l \tilde{\mathbf{S}}_j \right)^\top \mathbb{E}[\mathbf{M}] \right. \right. \\
&\quad \quad \left. \left. \times \left(\sum_{j=1}^{n-1} \sum_{l=1}^{j-1} \left(\prod_{p=1}^{l-1} (\mathbf{I} - \mathbf{S}_p) \right) \mathbf{S}_l \mathbf{S}_j \left(\sum_{i=1}^{n-j} \mathbf{t}_{n+1-i} \right) \right) \right] \right\| \\
&\leq \mathbb{E} \left[\sum_{j=1}^{2n} \sum_{l=1}^{j-1} \left(\prod_{p=1}^{l-1} \|\mathbf{I} - \tilde{\mathbf{S}}_p\| \right) \|\tilde{\mathbf{S}}_l\| \|\tilde{\mathbf{S}}_j\| \right] \mathbb{E}[\|\mathbf{M}\|] \left\| \sum_{i=1}^n \mathbf{S}_i \mathbf{t}_i \right\| \\
&\quad + \mathbb{E} \left[\left(\sum_{j=1}^{2n} \sum_{l=1}^{j-1} \left(\prod_{p=1}^{l-1} \|\mathbf{I} - \tilde{\mathbf{S}}_p\| \right) \|\tilde{\mathbf{S}}_l\| \|\tilde{\mathbf{S}}_j\| \right) \mathbb{E}[\|\mathbf{M}\|] \right. \\
&\quad \quad \left. \times \left(\sum_{j=n+1}^{2n-1} \sum_{l=1}^{j-1} \left(\prod_{p=1}^{l-1} \|\mathbf{I} - \tilde{\mathbf{S}}_p\| \right) \|\tilde{\mathbf{S}}_l\| \|\tilde{\mathbf{S}}_j\| \left\| \sum_{i=1}^{2n-j} \mathbf{t}_i \right\| \right) \right] \\
&\quad + \mathbb{E} \left[\left(\sum_{j=1}^{2n} \sum_{l=1}^{j-1} \left(\prod_{p=1}^{l-1} \|\mathbf{I} - \tilde{\mathbf{S}}_p\| \right) \|\tilde{\mathbf{S}}_l\| \|\tilde{\mathbf{S}}_j\| \right) \mathbb{E}[\|\mathbf{M}\|] \right. \\
&\quad \quad \left. \times \left(\sum_{j=1}^{n-1} \sum_{l=1}^{j-1} \left(\prod_{p=1}^{l-1} \|\mathbf{I} - \mathbf{S}_p\| \right) \|\mathbf{S}_l\| \|\mathbf{S}_j\| \left\| \sum_{i=1}^{n-j} \mathbf{t}_{n+1-i} \right\| \right) \right].
\end{aligned}$$

Now, we use that $\|\mathbf{M}\| \leq 1$, $\|\mathbf{I} - \widetilde{\mathbf{S}}_i\| \leq 1$, $\|\widetilde{\mathbf{S}}_i\| \leq \alpha L$ and $\|\tilde{\mathbf{t}}_i\| \leq \alpha G^*$:

$$\begin{aligned} & \left\| \mathbb{E} \left[\left(\sum_{j=1}^{2n} \sum_{l=1}^{j-1} \left(\prod_{p=1}^{l-1} (\mathbf{I} - \widetilde{\mathbf{S}}_p) \right) \widetilde{\mathbf{S}}_l \widetilde{\mathbf{S}}_j \right)^\top \mathbb{E}[\mathbf{M}] \mathbf{z} \right] \right\| \\ & \leq 4n^3 \alpha^4 L^3 G^* + 4n^4 \alpha^4 L^4 \mathbb{E} \left[\left\| \sum_{i=1}^{2n-j} \mathbf{t}_i \right\| \right] + n^4 \alpha^4 L^4 \mathbb{E} \left[\left\| \sum_{i=1}^{n-j} \mathbf{t}_{n+1-i} \right\| \right]. \end{aligned}$$

Using Lemma C.8, we get

$$\begin{aligned} & \left\| \mathbb{E} \left[\left(\sum_{j=1}^{2n} \sum_{l=1}^{j-1} \left(\prod_{p=1}^{l-1} (\mathbf{I} - \widetilde{\mathbf{S}}_p) \right) \widetilde{\mathbf{S}}_l \widetilde{\mathbf{S}}_j \right)^\top \mathbb{E}[\mathbf{M}] \mathbf{z} \right] \right\| \leq 4n^3 \alpha^4 L^3 G^* \\ & \quad + 15n^4 \alpha^5 L^4 G^* \sqrt{2n \log n}. \end{aligned}$$

Putting everything together,

$$\begin{aligned} & \left\| \mathbb{E} \left[\left(\prod_{l=0}^{k'} \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i^{\frac{K}{2}-l}) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}^{\frac{K}{2}-l}) \right) \right)^\top \right. \right. \\ & \quad \left. \left. \times \left(\prod_{l=0}^{k'-1} \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_i^{\frac{K}{2}-l}) \right) \left(\prod_{i=1}^n (\mathbf{I} - \mathbf{S}_{n-i+1}^{\frac{K}{2}-l}) \right) \right) \mathbf{z}^{\frac{K}{2}-k'} \right] \right\| \\ & \leq (28n\alpha^2 L G^* + 9\alpha^5 L^4 n^4 G^* \sqrt{2n \log n}) + (4n^3 \alpha^4 L^3 G^* + 15n^4 \alpha^5 L^4 G^* \sqrt{2n \log n}) \\ & \leq 32n\alpha^2 L G^* + 24\alpha^5 L^4 n^4 G^* \sqrt{2n \log n}. \end{aligned}$$

C.7 Proof of Theorem 6.11

Proof. We start off by defining the error term

$$\begin{aligned} \mathbf{r}^k &= \left(\sum_{i=1}^n \nabla f_i(\mathbf{x}_{i-1}^{2k-1}) - \sum_{i=1}^n \nabla f_i(\mathbf{x}_0^{2k-1}) \right) \\ & \quad + \left(\sum_{i=1}^n \nabla f_{n-i+1}(\mathbf{x}_{i-1}^{2k}) - \sum_{i=1}^n \nabla f_{n-i+1}(\mathbf{x}_0^{2k-1}) \right), \end{aligned}$$

where $k \in [K/2]$. This captures the difference between true gradients that the algorithms observes, and the gradients that a full step of gradient descent would have seen.

For two consecutive epochs of FLIPFLOP WITH INCREMENTAL GD, we have the following

inequality

$$\begin{aligned}
& \|\mathbf{x}_n^{2k} - \mathbf{x}^*\|^2 \\
&= \|\mathbf{x}_0^{2k-1} - \mathbf{x}^*\|^2 + \alpha^2 \left\| \sum_{i=1}^n \nabla f_i(\mathbf{x}_{i-1}^{2k-1}) + \sum_{i=1}^n \nabla f_{n-i+1}(\mathbf{x}_{i-1}^{2k}) \right\|^2 \\
&\quad - 2\alpha \left\langle \mathbf{x}_0^{2k-1} - \mathbf{x}^*, \sum_{i=1}^n \nabla f_i(\mathbf{x}_{i-1}^{2k-1}) + \sum_{i=1}^n \nabla f_{n-i+1}(\mathbf{x}_{i-1}^{2k}) \right\rangle \\
&= \|\mathbf{x}_0^{2k-1} - \mathbf{x}^*\|^2 - 2\alpha \left\langle \mathbf{x}_0^{2k-1} - \mathbf{x}^*, 2n \nabla F(\mathbf{x}_0^{2k-1}) \right\rangle \\
&\quad - 2\alpha \left\langle \mathbf{x}_0^{2k-1} - \mathbf{x}^*, \mathbf{r}^k \right\rangle + \alpha^2 \|2n \nabla F(\mathbf{x}_0^{2k-1}) + \mathbf{r}^k\|^2 \\
&\leq \|\mathbf{x}_0^{2k-1} - \mathbf{x}^*\|^2 - 4n\alpha \left[\frac{L\mu}{L+\mu} \|\mathbf{x}_0^{2k-1} - \mathbf{x}^*\|^2 + \frac{1}{L+\mu} \|\nabla F(\mathbf{x}_0^{2k-1})\|^2 \right] \\
&\quad - 2\alpha \left\langle \mathbf{x}_0^{2k-1} - \mathbf{x}^*, \mathbf{r}^k \right\rangle + \alpha^2 \|2n \nabla F(\mathbf{x}_0^{2k-1}) + \mathbf{r}^k\|^2 \\
&\leq \|\mathbf{x}_0^{2k-1} - \mathbf{x}^*\|^2 - 4n\alpha \left[\frac{L\mu}{L+\mu} \|\mathbf{x}_0^{2k-1} - \mathbf{x}^*\|^2 + \frac{1}{L+\mu} \|\nabla F(\mathbf{x}_0^{2k-1})\|^2 \right] \\
&\quad - 2\alpha \left\langle \mathbf{x}_0^{2k-1} - \mathbf{x}^*, \mathbf{r}^k \right\rangle + 8\alpha^2 n^2 \|\nabla F(\mathbf{x}_0^{2k-1})\|^2 + 2\alpha^2 \|\mathbf{r}^k\|^2 \\
&= \left(1 - 4n\alpha \frac{L\mu}{L+\mu} \right) \|\mathbf{x}_0^{2k-1} - \mathbf{x}^*\|^2 - \left(4n\alpha \frac{1}{L+\mu} - 8\alpha^2 n^2 \right) \|\nabla F(\mathbf{x}_0^{2k-1})\|^2 \\
&\quad - 2\alpha \left\langle \mathbf{x}_0^{2k-1} - \mathbf{x}^*, \mathbf{r}^k \right\rangle + 2\alpha^2 \|\mathbf{r}^k\|^2, \tag{C.29}
\end{aligned}$$

where the first inequality is due to Theorem 2.1.11 in Nesterov (2013) and the second one is simply $(a+b)^2 \leq 2a^2 + 2b^2$.

What remains to be done is to bound the two terms with $\mathbf{r}^k$ dependence. Firstly, we give a bound on the norm of $\mathbf{r}^k$:

$$\begin{aligned}
\|\mathbf{r}^k\| &= \left\| \left(\sum_{i=1}^n \nabla f_i(\mathbf{x}_{i-1}^{2k-1}) - \sum_{i=1}^n \nabla f_i(\mathbf{x}_0^{2k-1}) \right) \right. \\
&\quad \left. + \left(\sum_{i=1}^n \nabla f_{n-i+1}(\mathbf{x}_{i-1}^{2k}) - \sum_{i=1}^n \nabla f_{n-i+1}(\mathbf{x}_0^{2k-1}) \right) \right\| \\
&\leq \sum_{i=1}^n \|\nabla f_i(\mathbf{x}_{i-1}^{2k-1}) - \nabla f_i(\mathbf{x}_0^{2k-1})\| + \sum_{i=1}^n \|\nabla f_{n-i+1}(\mathbf{x}_{i-1}^{2k}) - \nabla f_{n-i+1}(\mathbf{x}_0^{2k-1})\|.
\end{aligned}$$

Next, we will use the smoothness assumption and bounded gradients property (Lemma C.1).

$$\|\mathbf{r}^k\| \leq L \sum_{i=1}^n \|\mathbf{x}_{i-1}^{2k-1} - \mathbf{x}_0^{2k-1}\| + L \sum_{i=1}^n \|\mathbf{x}_{i-1}^{2k} - \mathbf{x}_0^{2k-1}\|$$

$$\begin{aligned}
&\leq LG\alpha \sum_{i=1}^n i + LG\alpha \sum_{i=1}^n (n+i) \\
&= n(2n-1)\alpha GL.
\end{aligned}$$

Hence,

$$\|\mathbf{r}^k\|^2 \leq 4n^4 \alpha^2 G^2 L^2. \quad (\text{C.30})$$

For the $\mathbf{r}^k$ term, we need a more careful bound. Since the Hessian is constant for quadratic functions, we use $\mathbf{H}_i$ to denote the Hessian matrix of function $f_i(\cdot)$. We start off by using the definition of $\mathbf{r}^k$:

$$\begin{aligned}
\mathbf{r}^k &= \left(\sum_{i=1}^n \nabla f_i(\mathbf{x}_{i-1}^{2k-1}) - \sum_{i=1}^n \nabla f_i(\mathbf{x}_0^{2k-1}) \right) \\
&\quad + \left(\sum_{i=1}^n \nabla f_{n-i+1}(\mathbf{x}_{i-1}^{2k}) - \sum_{i=1}^n \nabla f_{n-i+1}(\mathbf{x}_0^{2k-1}) \right) \\
&= \sum_{i=1}^n \left(\nabla f_i(\mathbf{x}_{i-1}^{2k-1}) - \nabla f_i(\mathbf{x}_0^{2k-1}) \right) + \sum_{i=1}^n \left(\nabla f_{n-i+1}(\mathbf{x}_{i-1}^{2k}) - \nabla f_{n-i+1}(\mathbf{x}_0^{2k-1}) \right) \\
&= \sum_{i=1}^n \mathbf{H}_i (\mathbf{x}_{i-1}^{2k-1} - \mathbf{x}_0^{2k-1}) + \sum_{i=1}^n \mathbf{H}_{n-i+1} (\mathbf{x}_{i-1}^{2k} - \mathbf{x}_0^{2k-1}),
\end{aligned}$$

where we used the fact that for a quadratic function f with Hessian $\mathbf{H}$, we have that $\nabla f(x) - \nabla f(y) = \mathbf{H}(x - y)$. After that, we express $\mathbf{x}_{i-1}^{2k-1} - \mathbf{x}_0^{2k-1}$ and $\mathbf{x}_{i-1}^{2k} - \mathbf{x}_0^{2k-1}$ as sum of gradient descent steps:

$$\begin{aligned}
\mathbf{r}^k &= \sum_{i=1}^n \mathbf{H}_i \left(\sum_{j=1}^{i-1} -\alpha \nabla f_j(\mathbf{x}_{j-1}^{2k-1}) \right) \\
&\quad + \sum_{i=1}^n \mathbf{H}_{n-i+1} \left(\sum_{j=1}^n -\alpha \nabla f_j(\mathbf{x}_{j-1}^{2k-1}) + \sum_{j=1}^{i-1} -\alpha \nabla f_{n-j+1}(\mathbf{x}_{j-1}^{2k}) \right) \\
&= -\alpha \sum_{i=1}^n \mathbf{H}_i \left(\sum_{j=1}^{i-1} \nabla f_j(\mathbf{x}_{j-1}^{2k-1}) \right) - \alpha \sum_{i=1}^n \mathbf{H}_{n-i+1} \left(\sum_{j=1}^{i-1} \nabla f_{n-j+1}(\mathbf{x}_{j-1}^{2k}) \right) \\
&\quad - \alpha \sum_{i=1}^n \mathbf{H}_{n-i+1} \left(\sum_{j=1}^n \nabla f_j(\mathbf{x}_{j-1}^{2k-1}) \right) \\
&= -\alpha \sum_{i=1}^n \mathbf{H}_i \left(\sum_{j=1}^{i-1} \nabla f_j(\mathbf{x}_0^{2k-1}) \right) - \alpha \sum_{i=1}^n \mathbf{H}_{n-i+1} \left(\sum_{j=1}^{i-1} \nabla f_{n-j+1}(\mathbf{x}_0^{2k}) \right) \\
&\quad - \alpha \sum_{i=1}^n \mathbf{H}_{n-i+1} \left(\sum_{j=1}^n \nabla f_j(\mathbf{x}_0^{2k-1}) \right)
\end{aligned}$$

$$\begin{aligned}
& -\alpha \sum_{i=1}^n \mathbf{H}_i \left(\sum_{j=1}^{i-1} \nabla f_j(\mathbf{x}_{j-1}^{2k-1}) - f_j(\mathbf{x}_0^{2k-1}) \right) \\
& -\alpha \sum_{i=1}^n \mathbf{H}_{n-i+1} \left(\sum_{j=1}^{i-1} \nabla f_{n-j+1}(\mathbf{x}_{j-1}^{2k}) - \nabla f_{n-j+1}(\mathbf{x}_0^{2k}) \right) \\
& -\alpha \sum_{i=1}^n \mathbf{H}_{n-i+1} \left(\sum_{j=1}^n \nabla f_j(\mathbf{x}_{j-1}^{2k-1}) - \nabla f_j(\mathbf{x}_0^{2k-1}) \right) \\
& = -2\alpha \sum_{i=1}^n \mathbf{H}_i \left(\sum_{j=1}^n \nabla f_j(\mathbf{x}_0^{2k-1}) \right) + \alpha \sum_{i=1}^n \mathbf{H}_i \nabla f_i(\mathbf{x}_0^{2k-1}) \\
& -\alpha \sum_{i=1}^n \mathbf{H}_i \left(\sum_{j=1}^{i-1} \nabla f_j(\mathbf{x}_{j-1}^{2k-1}) - f_j(\mathbf{x}_0^{2k-1}) \right) \\
& -\alpha \sum_{i=1}^n \mathbf{H}_{n-i+1} \left(\sum_{j=1}^{i-1} \nabla f_{n-j+1}(\mathbf{x}_{j-1}^{2k}) - \nabla f_{n-j+1}(\mathbf{x}_0^{2k}) \right) \\
& -\alpha \sum_{i=1}^n \mathbf{H}_i \left(\sum_{j=1}^n \nabla f_j(\mathbf{x}_{j-1}^{2k-1}) - \nabla f_j(\mathbf{x}_0^{2k-1}) \right)
\end{aligned}$$

Next, we use the fact that $\sum_{j=1}^n \nabla f_j(\mathbf{x}) = n \nabla F(\mathbf{x})$. We will also again use the fact that for a quadratic function f with Hessian $\mathbf{H}$, we have that $\nabla f(x) - \nabla f(y) = \mathbf{H}(x - y)$:

$$\begin{aligned}
\mathbf{r}^k & = -2\alpha \sum_{i=1}^n \mathbf{H}_i (n \nabla F(\mathbf{x}_0^{2k-1})) + \alpha \sum_{i=1}^n \mathbf{H}_i (\nabla f_i(\mathbf{x}_0^{2k-1}) - \nabla f_i(\mathbf{x}^*)) \\
& + \alpha \sum_{i=1}^n \mathbf{H}_i \nabla f_i(\mathbf{x}^*) \\
& -\alpha \sum_{i=1}^n \mathbf{H}_i \left(\sum_{j=1}^{i-1} \nabla f_j(\mathbf{x}_{j-1}^{2k-1}) - f_j(\mathbf{x}_0^{2k-1}) \right) \\
& -\alpha \sum_{i=1}^n \mathbf{H}_{n-i+1} \left(\sum_{j=1}^{i-1} \nabla f_{n-j+1}(\mathbf{x}_{j-1}^{2k}) - \nabla f_{n-j+1}(\mathbf{x}_0^{2k}) \right) \\
& -\alpha \sum_{i=1}^n \mathbf{H}_i \left(\sum_{j=1}^n \nabla f_j(\mathbf{x}_{j-1}^{2k-1}) - \nabla f_j(\mathbf{x}_0^{2k-1}) \right) \\
& = -2\alpha \left(\sum_{i=1}^n \mathbf{H}_i \right)^2 (\mathbf{x}_0^{2k-1} - \mathbf{x}^*) + \alpha \sum_{i=1}^n \mathbf{H}_i^2 (\mathbf{x}_0^{2k-1} - \mathbf{x}^*) + \alpha \sum_{i=1}^n \mathbf{H}_i \nabla f_i(\mathbf{x}^*) \\
& -\alpha \sum_{i=1}^n \mathbf{H}_i \left(\sum_{j=1}^{i-1} \nabla f_j(\mathbf{x}_{j-1}^{2k-1}) - f_j(\mathbf{x}_0^{2k-1}) \right) \\
& -\alpha \sum_{i=1}^n \mathbf{H}_{n-i+1} \left(\sum_{j=1}^{i-1} \nabla f_{n-j+1}(\mathbf{x}_{j-1}^{2k}) - \nabla f_{n-j+1}(\mathbf{x}_0^{2k}) \right)
\end{aligned}$$

$$\begin{aligned}
& -\alpha \sum_{i=1}^n \mathbf{H}_i \left(\sum_{j=1}^n \nabla f_j(\mathbf{x}_{j-1}^{2k-1}) - \nabla f_j(\mathbf{x}_0^{2k-1}) \right) \\
& = \mathbf{a}^k + \mathbf{b}^k,
\end{aligned}$$

where the random variables $\mathbf{a}^k, \mathbf{b}^k$ as

$$\begin{aligned}
\mathbf{a}^k &:= -2\alpha \left(\sum_{i=1}^n \mathbf{H}_i \right)^2 (\mathbf{x}_0^{2k-1} - \mathbf{x}^*) + \alpha \sum_{i=1}^n \mathbf{H}_i^2 (\mathbf{x}_0^{2k-1} - \mathbf{x}^*) + \alpha \sum_{i=1}^n \mathbf{H}_i \nabla f_i(\mathbf{x}^*), \text{ and} \\
\mathbf{b}^k &:= -\alpha \sum_{i=1}^n \mathbf{H}_i \left(\sum_{j=1}^{i-1} \nabla f_j(\mathbf{x}_{j-1}^{2k-1}) - f_j(\mathbf{x}_0^{2k-1}) \right) \\
& \quad - \alpha \sum_{i=1}^n \mathbf{H}_{n-i+1} \left(\sum_{j=1}^{i-1} \nabla f_{n-j+1}(\mathbf{x}_{j-1}^{2k-1}) - \nabla f_{n-j+1}(\mathbf{x}_0^{2k-1}) \right) \\
& \quad - \alpha \sum_{i=1}^n \mathbf{H}_i \left(\sum_{j=1}^n \nabla f_j(\mathbf{x}_{j-1}^{2k-1}) - \nabla f_j(\mathbf{x}_0^{2k-1}) \right).
\end{aligned}$$

Again, using smoothness assumption and bounded gradients property (Lemma C.1) we get,

$$\|\mathbf{b}^k\| \leq 3\alpha^2 L^2 G n^3. \quad (\text{C.31})$$

Next, we decompose the inner product of $\mathbf{x}_0^{2k-1} - \mathbf{x}^*$ and $\mathbb{E}[\mathbf{r}^k]$ in Eq. (C.29):

$$\begin{aligned}
-2\alpha \langle \mathbf{x}_0^{2k-1} - \mathbf{x}^*, \mathbf{r}^k \rangle &= -2\alpha \langle \mathbf{x}_0^{2k-1} - \mathbf{x}^*, \mathbf{a}^k + \mathbf{b}^k \rangle \\
&= -2\alpha \langle \mathbf{x}_0^{2k-1} - \mathbf{x}^*, \mathbf{a}^k \rangle - 2\alpha \langle \mathbf{x}_0^{2k-1} - \mathbf{x}^*, \mathbf{b}^k \rangle
\end{aligned} \quad (\text{C.32})$$

For the first term in (C.32),

$$\begin{aligned}
-2\alpha \langle \mathbf{x}_0^{2k-1} - \mathbf{x}^*, \mathbf{a}^k \rangle &= 4\alpha^2 \left\langle \mathbf{x}_0^{2k-1} - \mathbf{x}^*, \left(\sum_{i=1}^n \mathbf{H}_i \right)^2 (\mathbf{x}_0^{2k-1} - \mathbf{x}^*) \right\rangle \\
& \quad - 2\alpha^2 \left\langle \mathbf{x}_0^{2k-1} - \mathbf{x}^*, \sum_{i=1}^n \mathbf{H}_i^2 (\mathbf{x}_0^{2k-1} - \mathbf{x}^*) \right\rangle \\
& \quad - 2\alpha^2 \left\langle \mathbf{x}_0^{2k-1} - \mathbf{x}^*, \sum_{i=1}^n \mathbf{H}_i \nabla f_i(\mathbf{x}^*) \right\rangle \\
&\leq 4\alpha^2 \left\langle \mathbf{x}_0^{2k-1} - \mathbf{x}^*, \left(\sum_{i=1}^n \mathbf{H}_i \right)^2 (\mathbf{x}_0^{2k-1} - \mathbf{x}^*) \right\rangle \\
& \quad - 2\alpha^2 \left\langle \mathbf{x}_0^{2k-1} - \mathbf{x}^*, \sum_{i=1}^n \mathbf{H}_i \nabla f_i(\mathbf{x}^*) \right\rangle
\end{aligned}$$

$$\begin{aligned}
&= 4\alpha^2 n^2 \|\nabla F(\mathbf{x}_0^{2k-1})\|^2 - 2\alpha^2 \left\langle \mathbf{x}_0^{2k-1} - \mathbf{x}^*, \sum_{i=1}^n \mathbf{H}_i \nabla f_i(\mathbf{x}^*) \right\rangle \\
&\leq 4\alpha^2 n^2 \|\nabla F(\mathbf{x}_0^{2k-1})\|^2 + 2\alpha^2 n L G \|\mathbf{x}_0^{2k-1} - \mathbf{x}^*\|. \tag{C.33}
\end{aligned}$$

For the second term in (C.32), we use Cauchy-Schwarz and Ineq. (C.31)

$$-2\alpha \langle \mathbf{x}_0^{2k-1} - \mathbf{x}^*, \mathbf{b}^k \rangle \leq 6\alpha^3 L^2 G n^3 \|\mathbf{x}_0^{2k-1} - \mathbf{x}^*\|. \tag{C.34}$$

Substituting (C.33) and (C.34) back to (C.32), we get

$$\begin{aligned}
-2\alpha \langle \mathbf{x}_0^{2k-1} - \mathbf{x}^*, \mathbf{r}^k \rangle &\leq 4\alpha^2 n^2 \|\nabla F(\mathbf{x}_0^{2k-1})\|^2 + 2\alpha^2 n L G \|\mathbf{x}_0^{2k-1} - \mathbf{x}^*\| \\
&\quad + 6\alpha^3 L^2 G n^3 \|\mathbf{x}_0^{2k-1} - \mathbf{x}^*\|. \tag{C.35}
\end{aligned}$$

Substituting (C.30) (C.35) back to (C.29), we finally get a recursion bound for one epoch:

$$\begin{aligned}
&\|\mathbf{x}_n^{2k} - \mathbf{x}^*\|^2 \tag{C.36} \\
&\leq \left(1 - 4n\alpha \frac{L\mu}{L + \mu}\right) \|\mathbf{x}_0^{2k-1} - \mathbf{x}^*\|^2 - \left(4n\alpha \frac{1}{L + \mu} - 8\alpha^2 n^2\right) \|\nabla F(\mathbf{x}_0^{2k-1})\|^2 \\
&\quad - 2\alpha \langle \mathbf{x}_0^{2k-1} - \mathbf{x}^*, \mathbf{r}^k \rangle + 2\alpha^2 \|\mathbf{r}^k\|^2 \\
&\leq \left(1 - 4n\alpha \frac{L\mu}{L + \mu}\right) \|\mathbf{x}_0^{2k-1} - \mathbf{x}^*\|^2 - \left(4n\alpha \frac{1}{L + \mu} - 8\alpha^2 n^2\right) \|\nabla F(\mathbf{x}_0^{2k-1})\|^2 \\
&\quad + 4\alpha^2 n^2 \|\nabla F(\mathbf{x}_0^{2k-1})\|^2 + 2\alpha^2 n L G \|\mathbf{x}_0^{2k-1} - \mathbf{x}^*\| + 6\alpha^3 L^2 G n^3 \|\mathbf{x}_0^{2k-1} - \mathbf{x}^*\| \\
&\quad + 8\alpha^4 n^4 G^2 L^2 \\
&= \left(1 - 4n\alpha \frac{L\mu}{L + \mu}\right) \|\mathbf{x}_0^{2k-1} - \mathbf{x}^*\|^2 - \left(4n\alpha \frac{1}{L + \mu} - 12\alpha^2 n^2\right) \|\nabla F(\mathbf{x}_0^{2k-1})\|^2 \\
&\quad + 2\alpha^2 n L G (1 + 3\alpha L n^2) \|\mathbf{x}_0^{2k-1} - \mathbf{x}^*\| + 8\alpha^4 n^4 G^2 L^2.
\end{aligned}$$

Next, we use the fact that $2ab \leq \lambda a^2 + b^2/\lambda$ (for any $\lambda > 0$) on the term $2\alpha^2 n L G (1 + 3\alpha L n^2) \|\mathbf{x}_0^{2k-1} - \mathbf{x}^*\|$ to get that

$$\begin{aligned}
2\alpha^2 n L G (1 + 3\alpha L n^2) \|\mathbf{x}_0^{2k-1} - \mathbf{x}^*\| &\leq \frac{(\alpha^2 n L G (1 + 3\alpha L n^2))^2}{n\alpha\mu} + n\alpha\mu \|\mathbf{x}_0^{2k-1} - \mathbf{x}^*\|^2 \\
&= \mu^{-1} \alpha^3 n L^2 G^2 (1 + 3\alpha L n^2)^2 + n\alpha \|\mathbf{x}_0^{2k-1} - \mathbf{x}^*\|^2.
\end{aligned}$$

Substituting this back we get,

$$\|\mathbf{x}_n^{2k} - \mathbf{x}^*\|^2$$

$$\begin{aligned}
&\leq \left(1 - 4n\alpha \frac{L\mu}{L+\mu}\right) \|\mathbf{x}_0^{2k-1} - \mathbf{x}^*\|^2 - \left(4n\alpha \frac{1}{L+\mu} - 12\alpha^2 n^2\right) \|\nabla F(\mathbf{x}_0^{2k-1})\|^2 \\
&\quad + \mu^{-1} \alpha^3 n L^2 G^2 (1 + 3\alpha L n^2)^2 + n\alpha \mu \|\mathbf{x}_0^{2k-1} - \mathbf{x}^*\|^2 + 8\alpha^4 n^4 G^2 L^2 \\
&\leq (1 - 2n\alpha\mu + n\alpha\mu) \|\mathbf{x}_0^{2k-1} - \mathbf{x}^*\|^2 - \left(4n\alpha \frac{1}{L+\mu} - 12\alpha^2 n^2\right) \|\nabla F(\mathbf{x}_0^{2k-1})\|^2 \\
&\quad + \mu^{-1} \alpha^3 n L^2 G^2 (1 + 3\alpha L n^2)^2 + 8\alpha^4 n^4 G^2 L^2 \quad (\text{Since } \mu \leq L) \\
&= (1 - n\alpha\mu) \|\mathbf{x}_0^{2k-1} - \mathbf{x}^*\|^2 - \left(4n\alpha \frac{1}{L+\mu} - 12\alpha^2 n^2\right) \|\nabla F(\mathbf{x}_0^{2k-1})\|^2 \\
&\quad + \mu^{-1} \alpha^3 n L^2 G^2 (1 + 3\alpha L n^2)^2 + 8\alpha^4 n^4 G^2 L^2
\end{aligned}$$

Now, substituting the values of α and the bound on K , we get that $4n\alpha \frac{1}{L+\mu} - 12\alpha^2 n^2 \geq 0$ and hence,

$$\|\mathbf{x}_n^{2k} - \mathbf{x}^*\|^2 \leq (1 - n\alpha\mu) \|\mathbf{x}_0^{2k-1} - \mathbf{x}^*\|^2 + \mu^{-1} \alpha^3 n L^2 G^2 (1 + 3\alpha L n^2)^2 + 8\alpha^4 n^4 G^2 L^2$$

Now, iterating this for $K/2$ epoch pairs, we get

$$\begin{aligned}
&\|\mathbf{x}_n^K - \mathbf{x}^*\|^2 \\
&\leq (1 - n\alpha\mu)^{K/2} \|\mathbf{x}_0^1 - \mathbf{x}^*\|^2 + \frac{K}{2} \mu^{-1} \alpha^3 n L^2 G^2 (1 + 3\alpha L n^2)^2 + 4K\alpha^4 n^4 G^2 L^2 \\
&\leq e^{-n\alpha\mu K/2} \|\mathbf{x}_0^1 - \mathbf{x}^*\|^2 + \frac{K}{2} \mu^{-1} \alpha^3 n L^2 G^2 (1 + 3\alpha L n^2)^2 + 4K\alpha^4 n^4 G^2 L^2 \\
&\leq e^{-n\alpha\mu K/2} \|\mathbf{x}_0^1 - \mathbf{x}^*\|^2 + K\mu^{-1} \alpha^3 n L^2 G^2 + 9K\mu^{-1} \alpha^5 n^5 L^4 G^2 + 4K\alpha^4 n^4 G^2 L^2 \\
&\quad (\text{Since } (1+a)^2 \leq 2+2a^2)
\end{aligned}$$

Substituting $\alpha = \frac{6 \log nK}{\mu nK}$ gives us the desired result. $\square$

C.8 Additional Experiments

Although our theoretical guarantees for FLIPFLOP only hold for quadratic objectives, we conjecture that FLIPFLOP might be able to improve the convergence performance for other classes of functions, whose Hessians are smooth near their minimizers. To see this, we also ran some experiments on 1-dimensional logistic regression. As we can see in Figure C.1, the convergence rates are very similar to those on quadratic functions. The data was synthetically generated such that the objective function becomes strongly convex and well conditioned near the minimizer. Note that logistic loss is not strongly convex on linearly

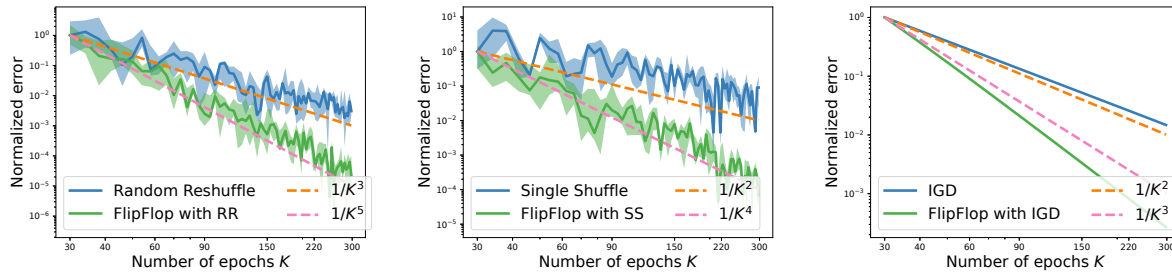


Figure C.1: Dependence of convergence rates on the number of epochs K for logistic regression. The figures show the median and inter-quartile range after 10 runs of each algorithm, with random initializations and random permutation seeds (note that IGD exhibits extremely small variance). We set $n = 800$, so that $n \gg K$ and hence the higher order terms of K dominate the convergence rates. Note that both the axes are in logarithmic scale.

separable data. Therefore, to make the loss strongly convex, we ensured that the data was not linearly separable. Essentially, the dataset was the following: all the datapoints were $z = \pm 1$, and their labels were $y = \mathbb{1}_{z>0}$ with probability $3/4$ and $y = \mathbb{1}_{z<0}$ with probability $1/4$. Framing this as an optimization problem, we have

$$\min_x F(x) := \mathbb{E} [-y \log(h(xz)) - (1 - y) \log(1 - h(xz))],$$

where $h(xz) = 1/(1 + e^{-xz})$. Note that $x = -\log 3$ is the minimizer of this function, which is helpful because we can use it to compute the exact error. Similar to the experiment on quadratic functions, n was set to 800 and step size was set in the same regime as in Theorems 6.9, 6.10, and 6.11.

D.1 Omitted proofs

D.1.1 Addition of pointers.

Lemma D.1. *There exists a 1-hidden layer feedforward, ReLU network, with $8d$ activations in the hidden layer and d neurons in the output layer that when given two d -dimensional binary vectors representing two non-negative integers, can output the binary vector representation of their sum, as long as the sum is less than 2^{d+1} .*

Proof. For the purpose of explaining this proof, we use the $\{0, 1\}^d$ binary representation of the integers, instead of the $\{\pm 1\}^d$ binary representation. However, since the conversion of a bit between the two representations can be done easily using simple affine transformation, the proof will also work for the $\{\pm 1\}^d$ binary representation.

Let the two integers be a, b and let $c := a + b$. We assume that $c < 2^d$. Further, let a_1 be the least significant bit of a , a_d the most significant, and a_i be the i -th most significant bit, and similarly for b and c . Further, let $a_{[i]}$ represent the integer formed by considering only the least i significant bits of a .

Note that c_i is only dependent on the least i bits of a and b , and not on the more significant bits of a or b . In particular, c_i only depends on $a_{[i]} + b_{[i]}$. Define $s := a_{[i]} + b_{[i]}$, and note that $c_i = s_i$. Further note that $s < 2^{i+1}$ and hence can be represented in $i + 1$ bits. Then, whenever $c_i = 1$, there can be two cases: $(s_{i+1} = 1, s_i = 1)$; or $(s_{i+1} = 0, s_i = 1)$. This can be equivalently written as $c_i = 1$ iff $s \in [2^{i-1}, 2^i - 1] \cup [3 \cdot 2^{i-1}, 2^{i+1} - 1]$. This can be computed by the following ReLU:

$$c_i = (\text{ReLU}(s - 2^{i-1} + 1) - \text{ReLU}(s - 2^{i-1})) + (\text{ReLU}(2^i - s) - \text{ReLU}(2^i - s - 1)) \\ - 1 + (\text{ReLU}(s - 3 \cdot 2^{i-1} + 1) - \text{ReLU}(s - 3 \cdot 2^{i-1})).$$

Thus, each bit of c can be computed using 6 neurons. Hence, computing the entire sum needs $8d$ activations, as to subtract the residual. $\square$

D.1.2 Non-linear functions as sum of sigmoids

Lemma D.2. *Consider an input of the form*

$$\mathbf{X} = \begin{bmatrix} \mathbf{e} & \mathbf{0} & \mathbf{x} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{p}_{2d+1} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{p}_1 & \mathbf{p}_{2:d} & \mathbf{0} & \mathbf{p}_{d+2:2d} & \mathbf{p}_{2d+1} & \mathbf{p}_{2d+2:3d} \\ 0 & 0_{2:d} & 1 & 0_{d+2:2d} & 0 & 0_{2d+2:3d} \end{bmatrix} \in \mathbb{R}^{N+d_x \times 3d}.$$

where d is chosen, N is the number of functions we encode and d_x is the dimension of $\mathbf{x}$. $\mathbf{e} = \mathbf{e}_j$ an indicator vector of the function we want to choose. Then there exists a transformer-based function block with 3 layers, m heads and dimensionality $O(d)$ such that

$$f(\mathbf{X}) = \begin{bmatrix} * & * & * & * & \sum_{i=1}^m c_{ji} \sigma(\mathbf{x}^T \mathbf{a}_{ji}) & * \\ \mathbf{0} & \mathbf{0} & \mathbf{x} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{p}_{2d+1} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{p}_1 & \mathbf{p}_{2:d} & \mathbf{0} & \mathbf{p}_{d+2:2d} & \mathbf{p}_{2d+1} & \mathbf{p}_{2d+2:3d} \\ 0 & 0_{2:d} & 1 & 0_{d+2:2d} & 0 & 0_{2d+2:3d} \end{bmatrix}$$

where $*$ denoted inconsequential values that will be ignored downstream.

Proof. The first thing we do is to move the $\mathbf{x}$ to the second row block, as follows:

$$\begin{aligned} \mathbf{X} &= \begin{bmatrix} \mathbf{e} & \mathbf{0} & \mathbf{x} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{p}_{2d+1} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{p}_1 & \mathbf{p}_{2:d} & \mathbf{0} & \mathbf{p}_{d+2:2d} & \mathbf{p}_{2d+1} & \mathbf{p}_{2d+2:3d} \\ 0 & 0_{2:d} & 1 & 0_{d+2:2d} & 0 & 0_{2d+2:3d} \end{bmatrix} \\ &\rightarrow \begin{bmatrix} \mathbf{e} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{x} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{p}_{2d+1} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{p}_1 & \mathbf{p}_{2:d} & \mathbf{0} & \mathbf{p}_{d+2:2d} & \mathbf{p}_{2d+1} & \mathbf{p}_{2d+2:3d} \\ 0 & 0_{2:d} & 1 & 0_{d+2:2d} & 0 & 0_{2d+2:3d} \end{bmatrix} \end{aligned}$$

This can be done using a ReLU feedforward layer that performs this using the last row of the input as the indicator bit for the column containing $\mathbf{x}$.

$$= \begin{bmatrix} * & * & \sigma(\mathbf{x}^T \mathbf{a}_{ji}) & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \end{bmatrix}$$

since $\mathbf{a}_{ji}^\top \mathbf{x} = \mathbf{a}_{ji}^\top \mathbf{x} - \log 3d - 1$ and thus $e^{\mathbf{a}_{ji}^\top \mathbf{x}} / (3d - 1 + e^{\mathbf{a}_{ji}^\top \mathbf{x}}) = \sigma(\mathbf{a}_{ji}^\top \mathbf{x})$ with a slight abuse of notation over the inner product $\mathbf{a}_{ji}^\top \mathbf{x}$ to account for the extra corrections bias term. Thus,

$$\mathbf{VX}_{\text{softmax}}((\mathbf{KX})^T(\mathbf{QX})) = \begin{bmatrix} * & * & c_{ji}\sigma(\mathbf{x}^T \mathbf{a}_{ji}) & * & * & * \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix}$$

By summing over all heads and adding the residual we get

$$\begin{bmatrix} * & * & \sum_{i=1}^m c_{ji}\sigma(\mathbf{x}^T \mathbf{a}_{ji}) & * & * & * \\ \mathbf{0} & \mathbf{0} & \mathbf{x} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{p}_{2d+1} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{p}_1 & \mathbf{p}_{2:d} & \mathbf{0} & \mathbf{p}_{d+2:2d} & \mathbf{p}_{2d+1} & \mathbf{p}_{2d+2:3d} \\ 0 & 0_{2:d} & 1 & 0_{d+2:2d} & 0 & 0_{2d+2:3d} \end{bmatrix}$$

Finally, we use an extra layer similarly to Theorem 8.4 to write the result in the desired output. Hence, we get

$$\begin{bmatrix} * & * & * & * & \sum_{i=1}^m c_{ji}\sigma(\mathbf{x}^T \mathbf{a}_{ji}) & * \\ \mathbf{0} & \mathbf{0} & \mathbf{x} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{p}_{2d+1} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{p}_1 & \mathbf{p}_{2:d} & \mathbf{0} & \mathbf{p}_{d+2:2d} & \mathbf{p}_{2d+1} & \mathbf{p}_{2d+2:3d} \\ 0 & 0_{2:d} & 1 & 0_{d+2:2d} & 0 & 0_{2d+2:3d} \end{bmatrix}$$

□

However, we have another way of controlling the input, which is by the size of each attention mechanism, that is directly controlled by the dimension d of the embedding.

Lemma D.3. Consider an input of the form

$$\mathbf{X} = \begin{bmatrix} \mathbf{x} & \dots & \mathbf{x} \\ 0 & \dots & 0 \\ \mathbf{1} - \mathbf{e}_1 & \dots & \mathbf{1} - \mathbf{e}_m \\ \mathbf{e}_1 & \dots & \mathbf{e}_m \end{bmatrix} \quad (\text{D.1})$$

where $\mathbf{e}_i$ is the one hot vector with 1 in the i -th position and $\mathbf{x} \in \mathbb{R}^d$. Let m be the number of sigmoids we need to represent a function, then there exists a one layer transformer with 1 head such that

$$\text{Attn}(\mathbf{X}) = \begin{bmatrix} \mathbf{x} & \dots & \mathbf{x} \\ \sigma(\mathbf{a}_1^\top \mathbf{x}) & \dots & \sigma(\mathbf{a}_m^\top \mathbf{x}) \\ \mathbf{1} - \mathbf{e}_1 & \dots & \mathbf{1} - \mathbf{e}_m \\ \mathbf{e}_1 & \dots & \mathbf{e}_m \end{bmatrix} \quad (\text{D.2})$$

Proof. Let

$$\mathbf{K} = \begin{bmatrix} \mathbf{0}^\top & 0 & \mathbf{0}^\top & \mathbf{e}_1^\top \\ \vdots & \vdots & \vdots & \vdots \\ \mathbf{0}^\top & 0 & \mathbf{0}^\top & \mathbf{e}_m^\top \end{bmatrix}, \mathbf{Q} = \begin{bmatrix} \mathbf{a}_1^\top & 0 & -C\mathbf{e}_1^\top & \mathbf{0}^\top \\ \vdots & \vdots & \vdots & \vdots \\ \mathbf{a}_m^\top & 0 & -C\mathbf{e}_m^\top & \mathbf{0}^\top \\ \mathbf{0}^\top & 0 & \mathbf{0}^\top & \mathbf{0}^\top \end{bmatrix} \quad (\text{D.3})$$

Hence,

$$\mathbf{KX} = \mathbf{I}_d, \mathbf{QX} = \begin{bmatrix} \mathbf{a}_1^\top \mathbf{x} & -C + \mathbf{a}_1^\top \mathbf{x} & \dots & -C + \mathbf{a}_1^\top \mathbf{x} \\ -C + \mathbf{a}_2^\top \mathbf{x} & \mathbf{a}_2^\top \mathbf{x} & \dots & -C + \mathbf{a}_2^\top \mathbf{x} \\ \vdots & \vdots & \vdots & \vdots \\ -C + \mathbf{a}_m^\top \mathbf{x} & -C + \mathbf{a}_m^\top \mathbf{x} & \dots & \mathbf{a}_m^\top \mathbf{x} \\ 0 & 0 & \dots & 0 \end{bmatrix} \quad (\text{D.4})$$

After applying softmax we get,

$$\sigma_s((\mathbf{KX})^\top \mathbf{QX}) \approx \begin{bmatrix} \sigma(\mathbf{a}_1^\top \mathbf{x}) & 0 & \dots & 0 \\ 0 & \sigma(\mathbf{a}_2^\top \mathbf{x}) & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & \sigma(\mathbf{a}_m^\top \mathbf{x}) \\ * & 0 & \dots & * \end{bmatrix},$$

for large enough C . Next we set

$$\mathbf{V} = \begin{bmatrix} \mathbf{0} & 0 & \mathbf{0} & 0 & \dots & 0 \\ \mathbf{0} & 0 & \mathbf{0} & c_1 & \dots & c_m \\ \mathbf{0} & 0 & \mathbf{0} & 0 & \dots & 0 \\ \mathbf{0} & 0 & \mathbf{0} & 0 & \dots & 0 \end{bmatrix}$$

thus resulting in

$$\mathbf{V}\mathbf{X} = \begin{bmatrix} 0 & 0 & \dots & 0 & 0 \\ c_1 & c_2 & \dots & c_m & 0 \\ 0 & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & 0 & 0 \end{bmatrix}$$

Hence, we get

$$\mathbf{V}\mathbf{X}_{\sigma_s}((\mathbf{K}\mathbf{X})^\top \mathbf{Q}\mathbf{X}) = \begin{bmatrix} 0 & \dots & 0 \\ c_1\sigma(\mathbf{a}_1^\top \mathbf{x}) & \dots & c_m\sigma(\mathbf{a}_m^\top \mathbf{x}) \\ 0 & \dots & 0 \\ \vdots & \vdots & \vdots \\ 0 & \dots & 0 \\ 0 & \dots & 0 \end{bmatrix},$$

and

$$\mathbf{X} + \mathbf{V}\mathbf{X}_{\sigma_s}((\mathbf{K}\mathbf{X})^\top \mathbf{Q}\mathbf{X}) = \begin{bmatrix} \mathbf{x} & \dots & \mathbf{x} \\ c_1\sigma(\mathbf{a}_1^\top \mathbf{x}) & \dots & c_m\sigma(\mathbf{a}_m^\top \mathbf{x}) \\ \mathbf{1} - \mathbf{e}_1 & \dots & \mathbf{1} - \mathbf{e}_m \\ \mathbf{e}_1 & \dots & \mathbf{e}_m \end{bmatrix}.$$

□

Corollary D.4. *Consider an input of the form*

$$\mathbf{X} = \begin{bmatrix} \mathbf{x} & \dots & \mathbf{0} \\ 0 & \dots & 0 \\ \mathbf{1} - \mathbf{e}_1 & \dots & \mathbf{1} - \mathbf{e}_m \\ \mathbf{e}_1 & \dots & \mathbf{e}_m \end{bmatrix} \tag{D.5}$$

where m is the number of sigmoids we use and $\mathbf{e}_i$ is an indicator vector and $\mathbf{x} \in \mathbb{R}^d$; then there exists a 3 layer transformer with 1 head such that

$$\text{Attn}(\mathbf{X}) = \begin{bmatrix} \sum_{i=1}^m \sigma(\mathbf{a}_i^\top \mathbf{x}) & \dots & \sum_{i=1}^m \sigma(\mathbf{a}_i^\top \mathbf{x}) \\ \mathbf{0} & \dots & \mathbf{0} \end{bmatrix} \quad (\text{D.6})$$

Proof. Given the input

$$\mathbf{X} = \begin{bmatrix} \mathbf{x} & \dots & \mathbf{0} \\ 0 & \dots & 0 \\ \mathbf{1} - \mathbf{e}_1 & \dots & \mathbf{1} - \mathbf{e}_m \\ \mathbf{e}_1 & \dots & \mathbf{e}_m \end{bmatrix}, \quad (\text{D.7})$$

we set the query and key matrices as follows:

$$\mathbf{K} = \mathbf{Q} = \begin{bmatrix} \mathbf{0}^\top & 0 & \mathbf{1} & \mathbf{1} \end{bmatrix}.$$

Then, we get

$$(\mathbf{KX})^\top \mathbf{QX} = \begin{bmatrix} d & \dots & d \\ \vdots & \dots & \vdots \\ d & \dots & d \end{bmatrix}.$$

Setting the value matrix to

$$\begin{bmatrix} d\mathbf{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix},$$

we get

$$\mathbf{VX}_{\text{softmax}}((\mathbf{KX})^\top \mathbf{QX}) = \begin{bmatrix} \mathbf{x} & \dots & \mathbf{x} \\ 0 & \dots & 0 \\ \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \dots & \mathbf{0} \end{bmatrix}.$$

Hence, the output of the attention layer is:

$$\mathbf{X} + \mathbf{V}\mathbf{X}\text{softmax}((\mathbf{K}\mathbf{X})^\top \mathbf{Q}\mathbf{X}) = \begin{bmatrix} 2\mathbf{x} & \dots & \mathbf{x} \\ 0 & \dots & 0 \\ \mathbf{1} - \mathbf{e}_1 & \dots & \mathbf{1} - \mathbf{e}_m \\ \mathbf{e}_1 & \dots & \mathbf{e}_m \end{bmatrix}.$$

Note that using the embeddings in the last rows and a feedforward network can be used to produce the following

$$\begin{bmatrix} \mathbf{x} & \dots & \mathbf{x} \\ 0 & \dots & 0 \\ \mathbf{1} - \mathbf{e}_1 & \dots & \mathbf{1} - \mathbf{e}_m \\ \mathbf{e}_1 & \dots & \mathbf{e}_m \end{bmatrix}.$$

Now, passing this into the transformer of Theorem D.3 will result in

$$\text{Attn}(\mathbf{X}) = \begin{bmatrix} \mathbf{x} & \dots & \mathbf{x} \\ c_1\sigma(\mathbf{a}_1^\top \mathbf{x}) & \dots & c_m\sigma(\mathbf{a}_m^\top \mathbf{x}) \\ \mathbf{1} - \mathbf{e}_1 & \dots & \mathbf{1} - \mathbf{e}_m \\ \mathbf{e}_1 & \dots & \mathbf{e}_m \end{bmatrix}. \quad (\text{D.8})$$

For the third layer, we set the key and query matrices as follows

$$\mathbf{K} = \mathbf{Q} = \begin{bmatrix} \mathbf{0}^\top & 0 & 1 & 1 \end{bmatrix}.$$

Then, we get

$$(\mathbf{K}\mathbf{X})^\top \mathbf{Q}\mathbf{X} = \begin{bmatrix} m & \dots & m \\ \vdots & \dots & \vdots \\ m & \dots & m \end{bmatrix}.$$

Setting the value matrix to

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & m & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix},$$

we get

$$\mathbf{V}\mathbf{X}_{\text{softmax}}((\mathbf{K}\mathbf{X})^\top \mathbf{Q}\mathbf{X}) = \begin{bmatrix} \mathbf{0} & \dots & \mathbf{0} \\ \sum_{i=1}^m c_i \sigma(\mathbf{a}_i^\top \mathbf{x}) & \dots & \sum_{i=1}^m c_i \sigma(\mathbf{a}_i^\top \mathbf{x}) \\ \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \dots & \mathbf{0} \end{bmatrix}.$$

Hence, the output of the attention layer is:

$$\mathbf{X} + \mathbf{V}\mathbf{X}_{\text{softmax}}((\mathbf{K}\mathbf{X})^\top \mathbf{Q}\mathbf{X}) = \begin{bmatrix} \mathbf{x} & \dots & \mathbf{x} \\ \sum_{i=1}^m c_i \sigma(\mathbf{a}_i^\top \mathbf{x}) & \dots & \sum_{i=1}^m c_i \sigma(\mathbf{a}_i^\top \mathbf{x}) \\ \mathbf{1} - \mathbf{e}_1 & \dots & \mathbf{1} - \mathbf{e}_m \\ \mathbf{e}_1 & \dots & \mathbf{e}_m \end{bmatrix}.$$

Finally, the feedforward layers can be used to move the results to the first row. $\square$

D.1.3 Matrix Transposition

Lemma D.5. Fix $\epsilon > 0$ and consider an input of the following form

$$\mathbf{X} = \left[\begin{array}{c|c|c|c|c} \mathbf{A} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{p}_{1:d} & \mathbf{p}_{1:d} & \mathbf{p}_{1:d} & \dots & \mathbf{p}_{1:d} \\ \mathbf{P}'_1 & \mathbf{P}'_2 & \mathbf{P}'_3 & \dots & \mathbf{P}'_d \end{array} \right].$$

where $\mathbf{A} \in \mathbb{R}^{d \times d}$; then there exists transformer-based function block with 4 layers, 1 head and dimensionality $r = 2d + 2 \log d = O(d)$ that outputs the following matrix

$$\mathbf{X} = \left[\begin{array}{c|c|c|c|c} \mathbf{A}' & \mathbf{A}' & \mathbf{A}' & \dots & \mathbf{A}' \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{p}_{1:d} & \mathbf{p}_{1:d} & \mathbf{p}_{1:d} & \dots & \mathbf{p}_{1:d} \\ \mathbf{P}'_1 & \mathbf{P}'_2 & \mathbf{P}'_3 & \dots & \mathbf{P}'_d \end{array} \right].$$

where $\mathbf{A}' = \mathbf{A}^\top + \epsilon \mathbf{M}$, for some $\|\mathbf{M}\| \leq 1$.

Proof. We can vectorize the matrix $\mathbf{A}$ into a d^2 dimensional vector using the attention mechanism, as shown in Eq. (D.9). Notice that once we have the matrix in this form we can implement its transpose with a fixed permutation of the columns of the matrix to get the vectorized form of $\mathbf{A}^\top$. Once we have the transpose in vector form, we matricize it back

to get the matrix transform using the attention mechanism. We explain the details of this process below:

Vectorization: We assume that the input is of the following form, where $\mathbf{A}$ is the matrix to be vectorized.

$$\mathbf{X} = \begin{bmatrix} \mathbf{A} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{p}_{1:d} & \mathbf{p}_{1:d} & \dots & \mathbf{p}_{1:d} \\ \mathbf{P}'_1 & \mathbf{P}'_2 & \dots & \mathbf{P}'_d \end{bmatrix}.$$

Here, $\mathbf{P}'_i$ represents a matrix of d columns, where each column is $\mathbf{p}_i$.

The first layer uses the $\mathbf{p}_{1:d}$ encodings to make d copies of the matrix $\mathbf{A}$, as follows:

$$\mathbf{X} = \begin{bmatrix} \mathbf{A} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{A} & \mathbf{A} & \dots & \mathbf{A} \\ \mathbf{p}_{1:d} & \mathbf{p}_{1:d} & \dots & \mathbf{p}_{1:d} \\ \mathbf{P}'_1 & \mathbf{P}'_2 & \dots & \mathbf{P}'_d \end{bmatrix}.$$

The feed forward part of the second layer then uses the encodings $\mathbf{p}'_i$ to vectorize the matrix in the second row block as follows:

$$\mathbf{X} = \begin{bmatrix} \mathbf{A} & \dots & \mathbf{0} \\ \begin{bmatrix} A_{(1,1)} & \dots & A_{(1,d)} \\ \mathbf{0} & \dots & \mathbf{0} \end{bmatrix} & \dots & \begin{bmatrix} A_{(d,1)} & \dots & A_{(d,d)} \\ \mathbf{0} & \dots & \mathbf{0} \end{bmatrix} \\ \mathbf{p}_{1:d} & \dots & \mathbf{p}_{1:d} \\ \mathbf{P}'_1 & \dots & \mathbf{P}'_d \end{bmatrix}. \quad (\text{D.9})$$

This is achieved, by explicitly defining a neural network that keeps the i -th row if the corresponding encoding is $\mathbf{P}'_i$ and place it in the $d + 1$ row.

Transposition in the vector form: Once we have the matrix vectorized as the second row block of the scratchpad, the following key and query matrices

$$\mathbf{K} = \begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I} \end{bmatrix}, \mathbf{Q} = \begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I} \\ \mathbf{0} & \mathbf{0} & \mathbf{I} & \mathbf{0} \end{bmatrix},$$

results in the head outputting the following, which is the vectorized form of $\mathbf{A}^\top$ (in the

second row block)

$$\mathbf{X}_{\text{softmax}}((\mathbf{K}\mathbf{X})^\top(\mathbf{Q}\mathbf{X})) = \begin{bmatrix} * & \dots & * \\ \begin{bmatrix} A_{(1,1)} & \dots & A_{(d,1)} \\ \mathbf{0} & \dots & \mathbf{0} \end{bmatrix} & \dots & \begin{bmatrix} A_{(1,d)} & \dots & A_{(d,d)} \\ \mathbf{0} & \dots & \mathbf{0} \end{bmatrix} \\ \mathbf{P}'_1 & \dots & \mathbf{P}'_d \\ \mathbf{p}_{1:d} & \dots & \mathbf{p}_{1:d} \end{bmatrix}.$$

Then, using the following value matrix gives

$$\mathbf{V} = \begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix},$$

$$\mathbf{V}\mathbf{X}_{\text{softmax}}((\mathbf{K}\mathbf{X})^\top(\mathbf{Q}\mathbf{X})) = \begin{bmatrix} \mathbf{0} & \dots & \mathbf{0} \\ \begin{bmatrix} A_{(1,1)} & \dots & A_{(d,1)} \\ \mathbf{0} & \dots & \mathbf{0} \end{bmatrix} & \dots & \begin{bmatrix} A_{(1,d)} & \dots & A_{(d,d)} \\ \mathbf{0} & \dots & \mathbf{0} \end{bmatrix} \\ \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \dots & \mathbf{0} \end{bmatrix},$$

Adding back the $\mathbf{X}$ (see (8.1)), results in

$$\begin{aligned} \mathbf{X} + \mathbf{V}\mathbf{X}_{\text{softmax}}((\mathbf{K}\mathbf{X})^\top(\mathbf{Q}\mathbf{X})) \\ = \begin{bmatrix} \mathbf{A} & \dots & \mathbf{0} \\ \begin{bmatrix} A_{(1,1)} & \dots & A_{(d,1)} \\ \mathbf{0} & \dots & \mathbf{0} \end{bmatrix} & \dots & \begin{bmatrix} A_{(1,d)} & \dots & A_{(d,d)} \\ \mathbf{0} & \dots & \mathbf{0} \end{bmatrix} \\ \mathbf{p}_{1:d} & \dots & \mathbf{p}_{1:d} \\ \mathbf{P}'_1 & \dots & \mathbf{P}'_d \end{bmatrix}. \end{aligned}$$

Using the feedforward layers and the encodings $\mathbf{P}'_{i'}$, we get

$$\mathbf{X} = \begin{bmatrix} \mathbf{A} & \dots & \mathbf{0} \\ \begin{bmatrix} A_{(1,1)} & \dots & A_{(d,1)} \\ \mathbf{0} & \dots & \mathbf{0} \end{bmatrix} & \dots & \begin{bmatrix} \mathbf{0} & \dots & \mathbf{0} \\ A_{(1,d)} & \dots & A_{(d,d)} \end{bmatrix} \\ \mathbf{p}_{1:d} & \dots & \mathbf{p}_{1:d} \\ \mathbf{P}'_1 & \dots & \mathbf{P}'_d \end{bmatrix}.$$

Using an attention layer and the first row of encodings, we get

$$\mathbf{X} = \begin{bmatrix} \mathbf{A}^\top & \dots & \mathbf{A}^\top \\ \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{P}_{1:d} & \dots & \mathbf{P}_{1:d} \\ \mathbf{P}'_1 & \dots & \mathbf{P}'_d \end{bmatrix}.$$

D.1.4 Matrix Multiplication by Linearizing the Softmax

We will show how we can implement matrix multiplication so that it will fit our unified template. To do so, we need to show for example for the result of $\mathbf{A}^\top \mathbf{B}$, where $\mathbf{A} \in \mathbb{R}^{k \times m}$ and $\mathbf{B} \in \mathbb{R}^{k \times n}$ with $k, m, n < d$ we can achieve the following:

$$\left[\begin{array}{cc|cc|c} \mathbf{A} & \mathbf{0} & \mathbf{B} & \mathbf{0} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{array} \right] \rightarrow \left[\begin{array}{cc|cc|cc} * & * & * & * & \mathbf{A}^\top \mathbf{B} & * \\ \hline \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{array} \right]$$

Lemma D.6. *Let $\mathbf{A} \in \mathbb{R}^{k \times m}$ and $\mathbf{B} \in \mathbb{R}^{k \times n}$; then for any $\epsilon > 0$ there exists a transformer-based function block with 2 layers, 1 head and dimensionality $r = O(d)$ that outputs the multiplication $\mathbf{A}^\top \mathbf{B} + \epsilon \mathbf{M}$, for some $\|\mathbf{M}\| \leq 1$.*

Corollary D.7. *Let $\mathbf{A} \in \mathbb{R}^{k \times m}$ and $\mathbf{B} \in \mathbb{R}^{k \times n}$; then for any $\epsilon > 0$ there exists a transformer-based function block with 2 layers, 1 head and dimensionality $r = O(d)$ that outputs the multiplication $\mathbf{B}^\top \mathbf{A} + \epsilon \mathbf{M}$, for some $\|\mathbf{M}\| \leq 1$.*

Corollary D.8. *Let $\mathbf{A} \in \mathbb{R}^{k \times m}$ and $\mathbf{B} \in \mathbb{R}^{k \times n}$; then for any $\epsilon > 0$ there exists a transformer-based function block with 2 layers, 1 head and dimensionality $r = O(d)$ that outputs the multiplication $\mathbf{B}^\top \mathbf{B} + \epsilon \mathbf{M}$, for some $\|\mathbf{M}\| \leq 1$.*

Corollary D.9. *Let $\mathbf{A} \in \mathbb{R}^{k \times m}$ and $\mathbf{B} \in \mathbb{R}^{k \times n}$; then for any $\epsilon > 0$ there exists a transformer-based function block with 2 layers, 1 head and dimensionality $r = O(d)$ that outputs the multiplication $\mathbf{A}^\top \mathbf{A} + \epsilon \mathbf{M}$, for some $\|\mathbf{M}\| \leq 1$.*

We will prove just the first of these results and the rest are a simple corollary of it.

Proof. Let $\mathbf{M} \in \mathbb{R}^{2d \times 2d}$, $\mathbf{A} \in \mathbb{R}^{k \times m}$ and $\mathbf{B} \in \mathbb{R}^{k \times n}$ be the following matrices:

$$\mathbf{M} = \begin{bmatrix} \mathbf{A} & \mathbf{0} & \mathbf{B} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix}.$$

The zeros pad the rows and columns to ensure that the matrix M is $2d \times 2d$. Then, consider the input matrix to be of the following form:

$$\mathbf{X} = \begin{bmatrix} \mathbf{M} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{1}\mathbf{1}^\top & \mathbf{0} \\ \mathbf{I} & \mathbf{0} & \mathbf{0} \\ & \mathbf{p}^{(1)} & \\ & \mathbf{p}^{(2)} & \\ \mathbf{0} & \mathbf{1}^T & \mathbf{0} \end{bmatrix}$$

where $\mathbf{1} \in \mathbb{R}^{2d}$ is the all ones vector. The identity matrix $\mathbf{I}$ and the all ones matrix $\mathbf{1}\mathbf{1}^\top$ are part of the design of the input and they are always fixed. For now we ignore the encodings and the last row, by setting the corresponding rows of the key, query and value weight matrices to be zero. These rows will be used to copy the output to the place that we want.

Focusing on the rest of the rows, we set the key and query weight matrices to be

$$\mathbf{K} = \mathbf{I}, \mathbf{Q} = \begin{bmatrix} c\mathbf{I} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & C\mathbf{I} \\ \mathbf{0} & \mathbf{I} & \mathbf{0} \end{bmatrix}, \mathbf{V} = \begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & ne^C \mathcal{D}_d \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix}$$

where $\mathcal{D}_d \in \mathbb{R}^{2d \times 2d}$ is the diagonal matrix with the first d diagonal elements 1, and the rest 0. Thus we have

$$(\mathbf{K}\mathbf{X})^\top \mathbf{Q}\mathbf{X} = \begin{bmatrix} \mathbf{M} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{1}\mathbf{1}^\top & \mathbf{0} \\ \mathbf{I} & \mathbf{0} & \mathbf{0} \end{bmatrix}^\top \begin{bmatrix} c\mathbf{M} & \mathbf{0} & \mathbf{0} \\ C\mathbf{I} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{1}\mathbf{1}^\top & \mathbf{0} \end{bmatrix} \quad (\text{D.10})$$

$$= \begin{bmatrix} c\mathbf{M}^\top \mathbf{M} & \mathbf{1}\mathbf{1}^\top & \mathbf{0} \\ C\mathbf{1}\mathbf{1}^\top & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix} \quad (\text{D.11})$$

Each of the first $2d$ columns above looks as follows

$$\begin{bmatrix} cz_{1i} & cz_{2i} & \dots & cz_{ni} & C\mathbf{1}^\top & \mathbf{0} \end{bmatrix}$$

After we apply the softmax σ_s per column, we get

$$\sigma_s(cz_{ij}) = \frac{e^{cz_{ij}}}{\sum_{j=1}^n e^{cz_{ij}} + n(e^C + 1)}$$

where $n = 2d$, z_{ij} is the (i, j) element of the matrix $\mathbf{M}^\top \mathbf{M}$. Let $\ell(\cdot)$ be the transformation above then we have

$$\begin{aligned} \mathbf{V}\mathbf{X}_{\text{softmax}}((\mathbf{K}\mathbf{X})^\top \mathbf{Q}\mathbf{X}) &= \begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} \\ ne^C \mathcal{D}_d & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \ell(c\mathbf{M}^\top \mathbf{M}) & * & * \\ * & * & * \\ * & * & * \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} \\ ne^C \mathcal{D}_d \ell(c\mathbf{M}^\top \mathbf{M}) & * & * \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix} \\ &\approx \begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{1}\mathbf{1}^\top + c\mathbf{M}^\top \mathbf{M} & * & * \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix} \end{aligned}$$

and by adding back the residual we have

$$\mathbf{X} = \begin{bmatrix} \mathbf{M} & \mathbf{0} & \mathbf{0} \\ \mathbf{1}\mathbf{1}^\top + c\mathbf{M}^\top \mathbf{M} & * & * \\ \mathbf{I} & \mathbf{0} & \mathbf{0} \end{bmatrix}$$

for small enough c and large enough C . This is because

$$\begin{aligned} ne^C \frac{e^{cx_{ij}}}{\sum_{j=1}^n e^{cx_{ij}} + n(e^C + 1)} &= e^{cx_{ij}} \frac{1}{1 + \sum_{j=1}^n e^{cx_{ij}-C-\log n} + n} \\ &= (1 + cx_{ij} + O((cx_{ij})^2))(1 - e^{cx_{ij}-C-\log n} + O(e^{2(cx_{ij}-C-\log n)})) \\ &= (1 + cx_{ij} + O((cx_{ij})^2))(1 - e^{cx_{ij}-C-\log n}) \\ &\approx (1 + cx_{ij}) \end{aligned}$$

We now use the feedforward layers to perform the following transform

$$\mathbf{X} = \begin{bmatrix} * & * & * \\ \mathbf{M}^\top \mathbf{M} & * & * \\ * & * & * \end{bmatrix}$$

$$= \begin{bmatrix} * & * & * & * & * \\ \mathbf{A}^\top \mathbf{A} & \mathbf{0} & \mathbf{A}^\top \mathbf{B} & \mathbf{0} & * \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & * \\ \mathbf{B}^\top \mathbf{A} & \mathbf{0} & \mathbf{B}^\top \mathbf{B} & \mathbf{0} & * \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & * \\ * & * & * & * & * \end{bmatrix}$$

Now if $\mathbf{p}^{(1)} = [\mathbf{0} \ \mathbf{0} \ \mathbf{p}_{2d+1:2d+n} \ \mathbf{0} \ \mathbf{0}]$ and $\mathbf{p}^{(2)} = [\mathbf{p}_{1:n} \ \mathbf{p}_{n+1:d} \ \mathbf{0} \ \mathbf{p}_{d+n+1:2d} \ \mathbf{p}_{2d:3d}]$ we can copy $\mathbf{A}^\top \mathbf{B}$ to the desired place using Theorem 8.3. $\square$

D.2 Error Analysis

In all of this section we assume that each element of the input matrix $\mathbf{X}$ has values v_i bounded by some constant G , i.e., $|v_i| \leq G$.

The error in the read/ write operation. The positional encodings as we have already mentioned have the following properties: $\mathbf{p}_i$ is an $\log(n)$ dimensional ± 1 vector which is the binary representation of i with -1 in the place of 0. Hence, we have $\mathbf{p}_i^\top \mathbf{p}_i = \log(n)$ and each $\mathbf{p}_i^\top \mathbf{p}_j < \log(n)$ for $i \neq j$.

Each time a copy is implemented from one column to another, we create a permutation matrix (a matrix of zeros and ones) which then multiplies the input matrix $\mathbf{X} \in \mathbb{R}^{d \times n}$ from the right and results in permutations of the column space. We thus focus on just one column of the $n \times n$ matrix that is created after we apply the softmax. Let $\mathbf{z}$ be this column of the matrix, ideally we want to output in one position 1 and in the rest 0. In the place that we want to output 1, say the a -th position, we have the inner product $\mathbf{z}_a = \mathbf{p}_i^\top \mathbf{p}_i$ for some $i \in [n]$. The rest of the elements in the same column would be $\mathbf{z}_b \leq \mathbf{p}_i^\top \mathbf{p}_j$ for $i \neq j$ and $a \neq b$. Then,

$$\begin{aligned} [\text{softmax}((\mathbf{KX})^\top \mathbf{QX})]_{i,i} &= \frac{e^{\lambda \mathbf{p}_i^\top \mathbf{p}_i}}{e^{\lambda \mathbf{p}_i^\top \mathbf{p}_i} + \sum_{j \neq i} e^{\lambda \mathbf{p}_i^\top \mathbf{p}_j}} \\ &= \frac{1}{1 + \sum_{j \neq i} e^{\lambda \mathbf{p}_i^\top \mathbf{p}_j} / e^{\lambda \mathbf{p}_i^\top \mathbf{p}_i}} \end{aligned}$$

Since $\lambda \mathbf{p}_i^\top \mathbf{p}_j < \lambda \mathbf{p}_i^\top \mathbf{p}_i - \lambda$ for $i \neq j$, we have that

$$[\text{softmax}((\mathbf{KX})^\top \mathbf{QX})]_{i,i} \geq \frac{1}{1 + ne^{-\lambda}}$$

$$\begin{aligned}
&\geq \frac{1}{1 + e^{\log n - \lambda}} \\
&\geq 1 - \frac{e^{\log n - \lambda}}{1 + e^{\log n - \lambda}} \\
&\geq 1 - e^{\log n - \lambda}
\end{aligned}$$

Thus, for $i \neq j$, $[\text{softmax}((\mathbf{KX})^\top \mathbf{QX})]_{i,j} \leq e^{\log n - \lambda}$. This implies that there exist ϵ_i , $i = 1, \dots, n$ such that

$$\begin{aligned}
\mathbf{z}_a &= 1 - \epsilon_a, \text{ for some } \epsilon_a \leq e^{\log n - \lambda} \\
\mathbf{z}_b &= \epsilon_b \text{ for } b \neq a \text{ and for some } \epsilon_b \leq e^{\log n - \lambda}
\end{aligned}$$

Hence, we have that

$$\mathbf{z} = \mathbf{z}^* + \boldsymbol{\varepsilon}$$

where $\mathbf{z}^*$ is the targeted vector and $\boldsymbol{\varepsilon}$ is the vector containing the errors ϵ_a, ϵ_b .

Now let $\mathbf{x}_i$ be the i -th row of the input matrix $\mathbf{X}$, then we have

$$\begin{aligned}
\mathbf{Xz} &= \mathbf{Xz}^* + \mathbf{X}\boldsymbol{\varepsilon} \\
&= \mathbf{Xz}^* + \begin{bmatrix} \langle \mathbf{x}_1, \boldsymbol{\varepsilon} \rangle \\ \vdots \\ \langle \mathbf{x}_d, \boldsymbol{\varepsilon} \rangle \end{bmatrix}
\end{aligned}$$

In the general case that all the columns will change, let $\mathbf{P} = \text{softmax}((\mathbf{KX})^\top \mathbf{QX})$ and $\mathbf{P}^*$ be the targeted matrix then we have that

$$\mathbf{XP} = \mathbf{XP}^* + \mathbf{XE}$$

where $\mathbf{E} = \begin{bmatrix} \epsilon_1 & \dots & \epsilon_n \end{bmatrix}$ is the matrix containing all the errors and so

$$\begin{aligned}
\|\mathbf{XP} - \mathbf{XP}^*\| &= \max_{1 \leq j \leq n} \sum_{i=1}^d |\langle \mathbf{x}_i, \boldsymbol{\varepsilon}_j \rangle| \\
&\leq Gn^2 d e^{\log n - \lambda} \\
&\leq e^{\log Gdn^3 - \lambda}
\end{aligned}$$

Thus, if $\lambda > \log \frac{Gdn^3}{\epsilon}$ we have that

$$\|\mathbf{XP} - \mathbf{XP}^*\| \leq \epsilon$$

The error in Matrix Multiplication . This error has already been calculated in Appendix D.1.4, however we explicitly define it here as follows:

$$\begin{aligned} ne^C \frac{e^{cx_{ij}}}{\sum_{j=1}^n e^{cx_{ij}} + n(e^C + 1)} &= e^{cx_{ij}} \frac{1}{1 + \sum_{j=1}^n e^{cx_{ij}-C-\log n} + n} \\ &= (1 + cx_{ij} + O((cx_{ij})^2))(1 - e^{cx_{ij}-C-\log n} + O(e^{2(cx_{ij}-C-\log n)})) \end{aligned}$$

Let $c = \frac{\epsilon_1}{C_1 G}$ for some constant C_1 and $C = \log \frac{C_2}{\epsilon_2}$ for some C_2 then we have

$$\begin{aligned} A &= ne^C \frac{e^{cx_{ij}}}{\sum_{j=1}^n e^{cx_{ij}} + n(e^C + 1)} \\ &= e^{cx_{ij}} \frac{1}{1 + \sum_{j=1}^n e^{cx_{ij}-C-\log n} + n} \\ &= \left(1 + cx_{ij} + \frac{\epsilon_1^2 x_{ij}^2}{G^2}\right) \left(1 - \frac{e^{cx_{ij}\epsilon_2}}{n} + \frac{e^{2cx_{ij}\epsilon_2}}{n^2}\right) \\ &= (1 + cx_{ij}) \left(1 - \frac{e^{cx_{ij}\epsilon_2}}{n} + \frac{e^{2cx_{ij}\epsilon_2}}{n^2}\right) + \frac{\epsilon_1^2 x_{ij}^2}{G^2} \left(1 - \frac{e^{cx_{ij}\epsilon_2}}{n} + \frac{e^{2cx_{ij}\epsilon_2}}{n^2}\right) \end{aligned}$$

Thus,

$$\begin{aligned} |A - (1 + cx_{ij})| &= \left| -(1 + cx_{ij}) \frac{e^{cx_{ij}\epsilon_2}}{n} + \frac{e^{2cx_{ij}\epsilon_2}}{n^2} + \frac{\epsilon_1^2 x_{ij}^2}{G^2} \left(1 - \frac{e^{cx_{ij}\epsilon_2}}{n} + \frac{e^{2cx_{ij}\epsilon_2}}{n^2}\right) \right| \\ &\leq \epsilon_1^2 \left(\frac{e^{\epsilon_1/C_1}\epsilon_2}{n} + 2 \frac{e^{2\epsilon_1/C_1}\epsilon_2^2}{n^2} \right) + \frac{e^{\epsilon_1/C_1}\epsilon_2}{n} \left(1 + \frac{\epsilon_1}{C_1}\right) \\ &\leq 4 \frac{e^{\epsilon_1/C_1}\epsilon_2}{n} \end{aligned}$$

Hence if $\epsilon_2 = \epsilon/4$ and $\epsilon_1 = C_1 \log(n\epsilon)$ we have that the total error is less than ϵ .

Function approximation. The error in Theorem 8.13 is an immediate consequence of Theorem 8.12 and it is proportional to $1/\sqrt{m}$, where m is the number of heads we are using.

Accumulation of error after T operations. Fix an $\epsilon > 0$ and assume that in the t -th iteration the input is $\mathbf{X}_t = \mathbf{X}_t^* + \epsilon_t \mathbf{M}_t$, where $\mathbf{X}_t^*$ is the ideal input $0 < \epsilon_t < \frac{t\epsilon}{T}$ and $\mathbf{M}_t$ is a matrix such that $\|\mathbf{M}_t\| \leq 1$, we will show that $\mathbf{X}_{t+1} = \mathbf{X}_{t+1}^* + \epsilon_{t+1} \mathbf{M}_{t+1}$, where $\mathbf{X}_{t+1}^*$ is the ideal input, $0 < \epsilon_{t+1} < \frac{(t+1)\epsilon}{T}$ and $\mathbf{M}_{t+1}$ is a matrix such that $\|\mathbf{M}_{t+1}\| \leq 1$.

- Matrix Multiplication with a matrix $\mathbf{A}$, $\|\mathbf{A}\| \leq 1^1$ will have the following result:

$$\mathbf{A}\mathbf{X}_t + \epsilon' = \mathbf{A}\mathbf{X}_t^* + \epsilon_t \mathbf{A}\mathbf{M}_t + \epsilon' \mathbf{M}' = \mathbf{X}_{t+1}^* + (\epsilon_t + \epsilon') \mathbf{M}_{t+1}$$

where ϵ' is controlled by the constants we use in the design of the function block and $\mathbf{M}_{t+1}$ is some matrix with $\|\mathbf{M}_{t+1}\| \leq 1$. If now $\epsilon' < \frac{\epsilon}{T}$, our claim follows.

- Read/Write operations will result to an error of

$$\mathbf{X}_t \mathbf{P} = \mathbf{X}_t \mathbf{P}^* + \epsilon' \mathbf{M}' = \mathbf{X}_t^* \mathbf{P}^* + \epsilon_t \mathbf{M}_t \mathbf{P}^* + \epsilon' \mathbf{M}'$$

Notice that as before, since $\|\mathbf{M}'\| \leq 1$ and $\|\mathbf{M}_t \mathbf{P}^*\| \leq 1$ and thus we have $\mathbf{X}_{t+1} = \mathbf{X}_t \mathbf{P} = \mathbf{X}_{t+1}^* + \epsilon_{t+1} \mathbf{M}_{t+1}$, where $\epsilon_{t+1} = \epsilon_t + \epsilon'$. Again if $\epsilon' \leq \frac{\epsilon}{T}$ the result follows.

- The result for function approximation follows in a similar way.

D.3 subleq is Turing Complete

In this section, we show that our slightly restricted version of the original SUBLEQ instruction (Mavaddat and Parhami, 1988) is indeed also Turing complete. To do this, we will utilize Minsky machines, which are also Turing complete. A Minsky machine comprises of registers and a list of instructions, where each instruction can be either of the following two instructions

- **add(a):** $\text{mem}[a] := \text{mem}[a] + 1$, go to the next instruction.
- **sub(a, n):** If $\text{mem}[a] == 0$, go to instruction n . Otherwise $\text{mem}[a] := \text{mem}[a] - 1$, go to the next instruction.

Given a program written in a language above, we translate it into an equivalent one written in our SUBLEQ language. For this, we initialize three fixed locations / registers c_{-1} , c_0 , and c_{+1} such that $\text{mem}[c_{-1}] := -1$, $\text{mem}[c_0] := 0$, and $\text{mem}[c_{+1}] := +1$; as well as an extra register $\text{mem}[b]$. We translate the program instruction-by-instruction. Assume that we have translated the first $i - 1$ instructions. Let $j - 1$ be the index of the last (translated) SUBLEQ instruction, that is, the index of the next SUBLEQ instruction will be j . Then, for the i -th instruction in the Minsky machine language, we translate it into our language as follows:

¹Notice that this can be assumed without loss of generality, since we can normalize all the errors with the maximum norm of a matrix to the power of T .

- Case 1, The i -th instruction of the Minsky machine program is $\text{add}(a)$. This is equivalent to $\text{SUBLEQ}(a, c_{-1}, j + 1)$, and hence the j instruction in our program will simply be $\text{SUBLEQ}(a, c_{-1}, j + 1)$.
- Case 2, The i -th instruction in the Minsky machine program is $\text{sub}(a, n)$. This would be equivalent to the sequence of the following 5 SUBLEQ instructions.

Algorithm D.1 Translation for $\text{sub}(a, n)$

Instr. j : $\text{SUBLEQ}(b, b, j + 1)$
 Instr. $j + 1$: $\text{SUBLEQ}(b, a, j + 3)$
 Instr. $j + 2$: $\text{SUBLEQ}(a, c_{+1}, j + 5)$
 Instr. $j + 3$: $\text{SUBLEQ}(a, c_0, n')$
 Instr. $j + 4$: $\text{SUBLEQ}(a, c_{+1}, j + 5)$

Here n' is the index of the translation of the n -th instruction of the Minsky machine program. This can be computed as a function of the number of add and sub instructions up to instruction n . The correctness of the above can be verified by considering the three cases: $\text{mem}[a] \geq 1$, $\text{mem}[a] \leq -1$, and $\text{mem}[a] = 0$.

D.4 Single Instruction Set

Each instruction consists of the following tuple: $(p_a, p_b, p_c, p_{\text{flag}}, p_m, p_p)$, and does the following

1. $\text{mem}[c] = f_m(\text{mem}[a], \text{mem}[b])$
2. if $\text{mem}[\text{flag}]_{(0,0)} \leq 0$ goto instruction p

Here, locations a, b , and c can contain either scalars, or d -dimensional vectors or $d \times d$ matrices, and $\text{mem}[\text{flag}]_{(0,0)}$ is the 1-st entry of $\text{mem}[\text{flag}]$ if it is a vector / matrix, else it is $\text{mem}[\text{flag}]$ if a scalar.

This can be implemented using the following steps (each may use a separate layer of transformer):

At the beginning of each iteration, the scratchpad starts with storing the pointer to the next instruction p_t .

1. Read the command $(p_a, p_b, p_c, p_{\text{flag}}, p_p, p_m)$ from the location to the scratchpad.

2. Copy the $d \times d$ data at locations a, b to the scratchpad memory *scratchMem* (assume the data is $d \times d$ even if actually scalar or vector, the f_m implementation will handle that)
3. Copy the data to the i -th function row block using the feed forward layer.
4. Once in the correct row block, $f_m(\text{mem}[a], \text{mem}[b])$ is computed
5. Feedforward layers copy back the data from i -th row block to the scratchpad memory *scratchMem*.
6. Write result from scratchpad memory to $\mathbf{p}_c$.
7. if $\text{mem}[\text{flag}]_{(0,0)} \leq 0$ store $\mathbf{p}_p$ in the scratchpad, else $\mathbf{p}_{t+1}$

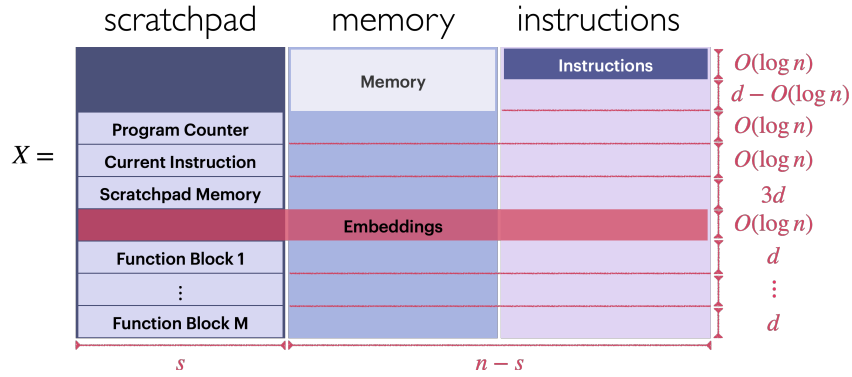


Figure D.1: The structure of input X

The structure of the input X is shown in Fig. D.1. It has n columns and $O(Md + \log n)$ rows. It is partitioned into 3 column blocks: the Scratchpad block, the Memory block, and the Instructions block. The Memory block is the storage and is the location where all the variables are stored. Each variable can be either a scalar, vector or matrix, as long as the number of rows in it are no larger than d . For example, if a variable is a $d \times d$ matrix, it is stored in d consecutive columns in the block, where each column has length d . The address of this variable is the index of its first column in the input X . The Instructions

block contains instructions, where each instruction is a vector of the form

$$\mathbf{c} = \begin{bmatrix} \mathbf{p}_a \\ \mathbf{p}_b \\ \mathbf{p}_c \\ \mathbf{p}_m \\ \mathbf{p}_{\text{flag}} \\ \mathbf{p}_p \\ d_h \\ d_w \\ b_{\text{mask}}^{(1)} \\ b_{\text{mask}}^{(2)} \\ b_{\text{mask}}^{(3)} \end{bmatrix},$$

which encodes the following logic:

$$\text{mem}[c] = f_m(\text{mem}[a], \text{mem}[b]) \quad ; \quad \text{if } \text{mem}[\text{flag}] \leq 0 \text{ goto instruction } p.$$

$\mathbf{p}_a, \mathbf{p}_b, \mathbf{p}_c, \mathbf{p}_p$, and $\mathbf{p}_{\text{flag}}$ are all binary ± 1 vectors that point to the locations a, b, c, p , and flag respectively. These are simply the binary representations of the integers a, b, c, p and flag, and hence have length $\log_2 n$ each. Similarly, $\mathbf{p}_m$ is the binary vector representation of the integer m , and hence has length $\log_2 M$, where M is the number of functions we implement. The b_{mask} is mask bit used while writing the output back to memory.

The scratchpad has s columns. The length s depends on the maximum number of columns needed by the function blocks to operate, and can be as low as $O(1)$ for scalar and vector functions, $O(d)$ for matrix functions, and can be as high as $O(d^2)$ if functions like matrix vectorization are one of the M functions. The Scratchpad consists of the following parts:

- The program counter is a row block with $\log_2 n$ rows and s columns and takes the form:

$$[\mathbf{p}_i \quad \mathbf{p}_i \quad \cdots \quad \mathbf{p}_i.]$$

This signifies that the current program counter points to the i -th instruction. Using this, the i -th instruction is read into all the s columns of ‘Current Instruction’ row block.

- The Current Instruction row block has $O(\log n)$ rows and s columns, and each column

initially contains the i -th instruction once it is read. Then, the instructions in each column are slightly modified depending on the column index, to read memory blocks pointed to in the instruction. The memory blocks are read into the ‘Scratchpad Memory’.

- The Scratchpad Memory is a temporary location where the data is first read into from the Memory column block, before it is moved to the correct function’s Function Block, using the function index encoding p_m in the instruction.
- The encodings row block has $O(\log n)$ rows and n columns, and is used to index every column in the input $\mathbf{X}$. It contains the binary ± 1 vector encodings of the column index for each column. The details of this row block are explained later.
- The Function Blocks are custom transformer blocks that can be added in a plug-n-play manner to the Unified Attention Based Computer depending on what ‘elementary’ functions the user wants the computer to have access to.

$$\mathbf{X} = \left[\begin{array}{cccc|ccc|ccc} \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{z}_{s+1} & \dots & \mathbf{z}_{m+s} & \left[\begin{array}{c} \mathbf{c}_{m+s+1} \\ \mathbf{0} \end{array} \right] & \dots & \left[\begin{array}{c} \mathbf{c}_n \\ \mathbf{0} \end{array} \right] \\ \hline \mathbf{p}_t & \mathbf{p}_t & \dots & \mathbf{p}_t & * & \dots & * & * & \dots & * \\ \hline \mathbf{c}_t^1 & \mathbf{c}_t^2 & \dots & \mathbf{c}_t^s & * & \dots & * & * & \dots & * \\ \hline \mathbf{z}_{a_t}^1 & \mathbf{z}_{a_t}^2 & \dots & \mathbf{z}_{a_t}^s & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{z}_{b_t}^1 & \mathbf{z}_{b_t}^2 & \dots & \mathbf{z}_{b_t}^s & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{z}_{c_t}^1 & \mathbf{z}_{c_t}^2 & \dots & \mathbf{z}_{c_t}^s & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{p}_{s+1} & \dots & \mathbf{p}_{m+s} & \mathbf{p}_{m+s+1} & \dots & \mathbf{p}_n \\ \mathbf{p}_1 & \mathbf{p}_2 & \dots & \mathbf{p}_s & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} \\ \hline \mathbf{f}_{1\text{mem}} & \dots & \dots & \dots & * & \dots & * & \dots & * & \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \dots & * & \\ \mathbf{f}_{M\text{mem}} & \dots & \dots & \dots & * & \dots & * & \dots & * & \end{array} \right]$$

D.4.1 Step 1

In this step, we need to copy the t -th instruction, pointed to by the program counter $\mathbf{p}_t$, to the scratchpad's Current Instruction block. We denote the instruction by $\mathbf{c}_t$ where

$$\mathbf{c}_t = \begin{bmatrix} \mathbf{p}_{a_t} \\ \mathbf{p}_{b_t} \\ \mathbf{p}_{c_t} \\ \mathbf{p}_{\text{flag}_t} \\ \mathbf{p}_{p_t} \\ \mathbf{p}_{m_t} \\ d_h \\ d_w \\ b_{\text{mask}}^{(1)} \\ b_{\text{mask}}^{(2)} \\ b_{\text{mask}}^{(3)} \end{bmatrix}$$

For this step, we only consider the following relevant subset of rows of the matrix $\mathbf{X}$:

$$\mathbf{X} = \left[\begin{array}{cccc|ccc|ccc} \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & * & * & \dots & \mathbf{c}_{m+s+1} & \dots & \mathbf{c}_n \\ \mathbf{p}_t & \mathbf{p}_t & \dots & \mathbf{p}_t & * & \dots & * & * & \dots & * \\ \mathbf{c}_t^1 & \mathbf{c}_t^2 & \dots & \mathbf{c}_t^s & * & \dots & * & * & \dots & * \\ \hline \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{p}_{s+1} & \dots & \mathbf{p}_{m+s} & \mathbf{p}_{m+s+1} & \dots & \mathbf{p}_n \end{array} \right]$$

The other rows will not be used or changed during this operation because we can simply set the corresponding rows of the $\mathbf{K}$, $\mathbf{V}$, $\mathbf{Q}$ matrices to 0 for all heads and setting the feed forward layers to also pass the corresponding rows unchanged.

At the beginning of execution of each command, the Current Instruction row block would be empty, so the input would look like

$$\mathbf{X} = \left[\begin{array}{cccc|ccc|ccc} * & * & \dots & * & * & * & \dots & \mathbf{c}_{m+s+1} & \dots & \mathbf{c}_n \\ \mathbf{p}_t & \mathbf{p}_t & \dots & \mathbf{p}_t & * & \dots & * & * & \dots & * \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} & \dots & * \\ \hline \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{p}_{s+1} & \dots & \mathbf{p}_{m+s} & \mathbf{p}_{m+s+1} & \dots & \mathbf{p}_n \end{array} \right]$$

Then, consider an attention head with the following $\mathbf{K}, \mathbf{Q}, \mathbf{V}$ matrices:

$$\mathbf{K} = \begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I} \end{bmatrix}, \mathbf{Q} = \begin{bmatrix} \mathbf{0} & \mathbf{I} & \mathbf{0} & \mathbf{0} \end{bmatrix}, \mathbf{V} = \begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix}$$

This will result in

$$\mathbf{X} = \left[\begin{array}{cccc|ccc|cccc} * & * & \dots & * & * & * & \dots & & \mathbf{c}_{m+s+1} & \dots & \mathbf{c}_n \\ \mathbf{p}_t & \mathbf{p}_t & \dots & \mathbf{p}_t & * & \dots & * & & * & \dots & * \\ \mathbf{c}_t & \mathbf{c}_t & \dots & \mathbf{c}_t & * & \dots & * & & * & \dots & * \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{p}_{s+1} & \dots & \mathbf{p}_{m+s} & & \mathbf{p}_{m+s+1} & \dots & \mathbf{p}_n \end{array} \right].$$

We apply Theorem D.1 on the row blocks

$$\left[\begin{array}{cccc|ccc|ccc} \mathbf{c}_t & \mathbf{c}_t & \dots & \mathbf{c}_t & * & \dots & * & * & \dots & * \\ \mathbf{p}_1 & \mathbf{p}_2 & \dots & \mathbf{p}_s & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} \end{array} \right]$$

to construct feedforward layers that convert $\mathbf{c}_t$ to $\mathbf{c}_t^i$, where

$$\mathbf{c}_t^i = \left[\begin{array}{c} \mathbf{p}_{a_t+i} \\ \mathbf{p}_{b_t+i-d} \\ \mathbf{p}_{c_t+i-2d} \\ \mathbf{p}_{\text{flag}_t} \\ \mathbf{p}_{p_t} \\ \mathbf{p}_{m_t} \\ d_h \\ d_w \\ b_{\text{mask}}^{(1)} = 1_{(i \leq d_w)} \\ b_{\text{mask}}^{(2)} = 1_{(i > d)} + 1_{(i \leq d+d_w)} - 1 \\ b_{\text{mask}}^{(3)} = 1_{(i > 2d)} + 1_{(i \leq 2d+d_w)} - 1 \end{array} \right].$$

Note that the last three elements can be created using the following ReLU:

$$\begin{aligned} b_{\text{mask}}^{(1)} &= \text{ReLU}(2d + d_w - i + 1) - \text{ReLU}(2d + d_w - i) \\ b_{\text{mask}}^{(2)} &= \text{ReLU}(i - d) - \text{ReLU}(i - d - 1) \\ &\quad + \text{ReLU}(d + d_w - i + 1) - \text{ReLU}(d + d_w - i) - 1 \end{aligned}$$

$$b_{\text{mask}}^{(3)} = \text{ReLU}(i - 2d) - \text{ReLU}(i - 2d - 1) \\ + \text{ReLU}(2d + d_w - i + 1) - \text{ReLU}(2d + d_w - i) - 1.$$

At the end of this step, we get the following:

$$\mathbf{X} = \left[\begin{array}{cccc|ccc|ccc} \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & * & * & \dots & & \mathbf{c}_{m+s+1} & \dots & \mathbf{c}_n \\ \mathbf{p}_t & \mathbf{p}_t & \dots & \mathbf{p}_t & * & \dots & * & & * & \dots & * \\ \mathbf{c}_t^0 & \mathbf{c}_t^1 & \dots & \mathbf{c}_t^s & * & \dots & * & & * & \dots & * \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{p}_{s+1} & \dots & \mathbf{p}_{m+s} & & \mathbf{p}_{m+s+1} & \dots & \mathbf{p}_n \end{array} \right],$$

D.4.2 Step 2

Use three heads, one each for $\mathbf{p}_a$, $\mathbf{p}_b$ and $\mathbf{p}_c$.

Using the vectors $\mathbf{p}_{a_t+i}$, $\mathbf{p}_{b_t+i-d}$, and $\mathbf{p}_{c_t+i-2d}$ we copy the data (using one head each and a similar technique as last step) to get the following in the Scratchpad Memory:

$$\left[\begin{array}{ccccccccc|ccc|ccc} \mathbf{z}_{a_t} & \dots & \mathbf{z}_{a_t+d} & * & \dots & * & * & \dots & * & * & \dots & * & * & \dots & * \\ * & \dots & * & \mathbf{z}_{b_t} & \dots & \mathbf{z}_{b_t+d} & * & \dots & * & * & \dots & * & * & \dots & * \\ * & \dots & * & * & \dots & * & \mathbf{z}_{c_t} & \dots & \mathbf{z}_{c_t+s-2d} & * & \dots & * & * & \dots & * \end{array} \right]$$

Using the mask bits at the end of $\mathbf{c}_t^i$, we get

$$\left[\begin{array}{ccccccccc|ccc|ccc} \mathbf{z}_{a_t} & \dots & \mathbf{z}_{a_t+d_w-1} & \mathbf{0} & \mathbf{z}_{b_t} & \dots & \mathbf{z}_{b_t+d_w-1} & \mathbf{0} & \mathbf{z}_{c_t} & \dots & \mathbf{z}_{c_t+d_w-1} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \dots \\ \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \dots \\ \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \dots \end{array} \right] \quad (\text{D.12})$$

$$\mathbf{z}_i[1:d] = \text{ReLU}(\mathbf{z}_i[1:d] - C(1 - b_{\text{mask}}^{(1)})\mathbf{1}) - \text{ReLU}(-\mathbf{z}_i[1:d] - C(1 - b_{\text{mask}}^{(1)})\mathbf{1}) \\ + \text{ReLU}(\mathbf{z}_i[d+1:2d] - C(1 - b_{\text{mask}}^{(2)})\mathbf{1}) \\ - \text{ReLU}(-\mathbf{z}_i[d+1:2d] - C(1 - b_{\text{mask}}^{(1)})\mathbf{1}) \\ + \text{ReLU}(\mathbf{z}_i[2d+1:3d] - C(1 - b_{\text{mask}}^{(1)})\mathbf{1}) \\ - \text{ReLU}(-\mathbf{z}_i[2d+1:3d] - C(1 - b_{\text{mask}}^{(1)})\mathbf{1}),$$

$$\mathbf{z}_i[d+1:3d] = \mathbf{0},$$

where C is a large positive constant.

Using the same mask bits, we also mask the row containing the output data pointers for c :

$$\left[\begin{array}{cccccccccccc|cccc|cccc} 0 & \dots & 0 & 0 & \dots & 0 & \mathbf{p}_{c_t} & \dots & \mathbf{p}_{c_t+d_w-1} & 0 & \dots & 0 & 0 & \dots & 0 & 0 & \dots & 0 \end{array} \right] \quad (\text{D.13})$$

D.4.3 Step 3

The following feedforward ReLU layer can move the data to the correct function blocks:

$$\begin{aligned} \mathbf{f}_{k\text{mem}}[1 : d_h] = & (\text{ReLU}(\mathbf{z}[1 : d_h] - C((1 - b_{\text{mask}}^{(1)} - b_{\text{mask}}^{(2)})\mathbf{1} + \log M - \mathbf{p}_k^\top \mathbf{p}_m)) \\ & - \text{ReLU}(-\mathbf{z}[1 : d_h] - C((1 - b_{\text{mask}}^{(1)} - b_{\text{mask}}^{(2)})\mathbf{1} + \log M - \mathbf{p}_k^\top \mathbf{p}_m))), \end{aligned}$$

where C is a large positive constant.

D.4.4 Step 4

Each of the M functions have their own attention heads, which are constructed to be copies of their transformer based function blocks. The results after the attention are written back into their respective row blocks. Since the row blocks are separate, the feedforward layers of each of the transformer based function blocks also work in parallel to store the final results in the respective row blocks.

D.4.5 Step 5

Similar to Step 3 we use the following feedforward ReLU layer to move the data from the function block back into the scratchpad memory

$$\begin{aligned} \mathbf{z}[1 : d_h] = & \mathbf{z}[1 : d_h] \\ & + \sum_{k=1}^M \left(\text{ReLU}((\mathbf{f}_{k\text{mem}}[1 : d_h] - \mathbf{z}[1 : d_h]) - C((1 - b_{\text{mask}}^{(3)})\mathbf{1} + \log M - \mathbf{p}_k^\top \mathbf{p}_m)) \right. \\ & \left. - \text{ReLU}(-(\mathbf{f}_{k\text{mem}}[1 : d_h] - \mathbf{z}[1 : d_h]) - C((1 - b_{\text{mask}}^{(3)})\mathbf{1} + \log M - \mathbf{p}_k^\top \mathbf{p}_m)) \right), \end{aligned}$$

where C is a large positive constant.

D.4.6 Step 6

For this step we focus on the encoding row block, memory storage row block and the following rows in the input (see (D.13), (D.12)):

$$\left[\begin{array}{c|c|c|c} 0 & \dots & 0 & 0 & \dots & 0 & 0 & \dots & 0 & 0 & \dots & 0 & \mathbf{z}_{s+1} & \dots & \mathbf{z}_{m+s} & \left[\begin{array}{c} \mathbf{c}_{m+s+1} \\ 0 \end{array} \right] & \dots & \left[\begin{array}{c} \mathbf{c}_n \\ 0 \end{array} \right] \\ \hline 0 & \dots & 0 & 0 & \dots & 0 & \mathbf{z}_{c_t}^{\text{new}} & \dots & \mathbf{z}_{c_t+d_w}^{\text{new}} & 0 & \dots & 0 & 0 & \dots & 0 & 0 & \dots & 0 \\ \hline 0 & \dots & 0 & 0 & \dots & 0 & \mathbf{p}_{c_t} & \dots & \mathbf{p}_{c_t+d_w} & 0 & \dots & 0 & 0 & \dots & 0 & 0 & \dots & 0 \\ \hline 0 & \dots & 0 & 0 & \dots & 0 & 0 & \dots & 0 & 0 & \dots & 0 & \mathbf{p}_s & \dots & \mathbf{p}_{m-1} & \mathbf{p}_m & \dots & \mathbf{p}_{n-1} \end{array} \right]$$

We set the Key and Query weight matrices as follows:

$$\mathbf{K} = \mathbf{Q} = \begin{bmatrix} 0 \\ 0 \\ \mathbf{I} \\ \mathbf{I} \end{bmatrix}.$$

$$\mathbf{V} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ \mathbf{I} & \mathbf{I} & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\mathbf{V}\mathbf{X}\text{softmax}((\mathbf{K}\mathbf{X})^\top \mathbf{Q}\mathbf{X}) =$$

$$\left[\begin{array}{ccccc|ccccc} \dots & \mathbf{0} & \dots & \mathbf{0} & \dots & \dots & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} & \dots \\ \hline \dots & \frac{d_{c_t}^{\text{new}} + d_{c_t}}{2} & \dots & \frac{d_{c_t+d_w}^{\text{new}} + d_{c_t+d_w}}{2} & \dots & \dots & d_{c_t-1} & \frac{d_{c_t}^{\text{new}} + d_{c_t}}{2} & \dots & \frac{d_{c_t+d_w}^{\text{new}} + d_{c_t+d_w}}{2} & d_{c_t+d_w+1} & \dots \\ \hline \dots & \mathbf{0} & \dots & \mathbf{0} & \dots & \dots & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} & \dots \\ \hline \dots & \mathbf{0} & \dots & \mathbf{0} & \dots & \dots & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} & \dots \end{array} \right]$$

Finally, we use the feedforward layers similar to the proof of Theorem 8.4 to write back $[d_{c_t}^{\text{new}} \dots d_{c_t+d_w}^{\text{new}}]$ to the correct rows.

D.4.7 Step 7

This step is identical to Section 8.4.3.

D.5 Calculator

Lemma D.10. *Given two constants $\epsilon, \delta \in [0, 1]$, there exists a 1 hidden layer neural network f with threshold activation and d activations in the hidden layer, such that*

$$\forall x \in [-C, -\delta] \cup [\delta, C], \left| f(x) - \frac{1}{x} \right| \leq \epsilon,$$

as long as $d = \Omega(\frac{\log(1/(\epsilon\delta))}{\epsilon\delta} + \log C)$.

Proof. We partition $[\delta, C]$ into the following intervals

$$[\delta, \delta(1 + \epsilon\delta)), [\delta(1 + \epsilon\delta), \delta(1 + \epsilon\delta)(1 + \epsilon\delta(1 + \epsilon\delta))] \dots, [a_i, a_i(1 + \epsilon a_i)), \dots,$$

that is, if an interval begins at a , then it ends at $a(1 + \epsilon a)$. Note that for any point $x \in [a_i, a_i(1 + \epsilon a_i))$

$$\begin{aligned} \left| \frac{1}{x} - \frac{1}{a_i} \right| &= \frac{1}{a_i} - \frac{1}{x} \\ &< \frac{1}{a_i} - \frac{1}{a_i(1 + \epsilon a_i)} \\ &= \frac{\epsilon}{1 + \epsilon a_i} < \epsilon. \end{aligned}$$

Hence two output activations of the form $\frac{1}{a_i}1_{x \geq a_i} - \frac{1}{a_i}1_{x < a_i(1+\epsilon a_i)}$ can be used to approximate $\frac{1}{x}$ in $[a_i, a_i(1 + \epsilon a_i))$.

Thus, all that remains is to compute the number of such intervals, and using that we get the number of output activations in the hidden layer. Towards that end, if the i -th interval begins at a_i ,

$$a_i = a_{i-1}(1 + \epsilon a_{i-1}) \geq a_{i-1}(1 + \epsilon \delta) = \delta(1 + \epsilon \delta)^{i-2}.$$

Hence,

$$\forall i \geq 2 + \frac{\log 1/(\epsilon \delta)}{\log(1 + \epsilon \delta)}, a_i \geq \frac{1}{\epsilon}.$$

Noting that $\log(1 + \epsilon \delta) > \frac{\epsilon \delta}{2}$ for $\epsilon, \delta \in [0, 1]$, we get that

$$\forall i \geq 2 + \frac{2 \log 1/(\epsilon \delta)}{\epsilon \delta}, a_i \geq 1.$$

Once we have that $a_i \geq \frac{1}{\epsilon}$, the number of further partitions needed to reach C would be $O(\log C)$ as shown below:

$$a_j = a_{j-1}(1 + \epsilon a_{j-1}) \geq a_{j-1} \left(1 + \epsilon \frac{1}{\epsilon}\right) = 2a_{j-1}.$$

Hence, the total number of partitions needed is $O(\frac{\log(1/(\epsilon \delta))}{\epsilon \delta} + \log C)$.

We can similarly approximate $1/x$ on $[-C, -\delta]$ with the same number of output activations. $\square$

Lemma D.11. *Given $\epsilon \in [0, 1]$, there exists a 1 hidden layer neural network f with threshold activation and d activations in the hidden layer, such that*

$$\forall x \in [0, C], |f(x) - \sqrt{x}| \leq \epsilon,$$

as long as $d = \Omega(\frac{\sqrt{C}}{\epsilon})$.

Proof. We partition $[0, C]$ into the following intervals

$$[0, \epsilon^2), [\epsilon^2, 4\epsilon^2), \dots, [i^2\epsilon^2, (i+1)^2\epsilon^2), \dots$$

Note that for any point $x \in [i^2\epsilon^2, (i+1)^2\epsilon^2)$

$$|\sqrt{x} - \sqrt{i^2\epsilon^2}| < \sqrt{(i+1)^2\epsilon^2} - \sqrt{i^2\epsilon^2} = \epsilon.$$

Hence two output activations of the form $i\epsilon 1_{x \geq i^2\epsilon^2} - i\epsilon 1_{x < (i+1)^2\epsilon^2}$ can be used to approximate $\sqrt{x}$ in $[i^2\epsilon^2, (i+1)^2\epsilon^2]$.

Thus, all that remains is to compute the number of such intervals, and using that we get the number of output activations in the hidden layer. It is easy to see that the total number of intervals needed would be $\frac{\sqrt{C}}{\epsilon}$. $\square$

E.1 Linear Algebra

In this part we mention some useful properties of the Kronecker and Khatri-Rao products, as well as some basic results from linear algebra.

E.1.1 Multiplication of matrices & full rank

Fact E.1. Let $\mathbf{A} \in \mathbb{R}^{n \times m}$ and $\mathbf{B} \in \mathbb{R}^{m \times l}$ be two full rank matrices, with $n < m$ and $l < m$. If the nullspace of $\mathbf{A}$ doesn't have intersection with the column space of $\mathbf{B}$ then $\text{rank}(\mathbf{AB}) = \min\{n, l\}$.

Remark E.2. We note here, that the above is a necessary and sufficient condition for the multiplication of two matrices to be full rank. The same argument can also be made through the left nullspace and the row space.

E.1.2 Some products & their properties

Definition E.3 (Kronecker product). The Kronecker product of two matrices, which we denote with $\mathbf{A} \otimes \mathbf{B}$, where $\dim(\mathbf{A}) = m \times n$ and $\dim(\mathbf{B}) = p \times q$ is defined to be the block matrix

$$\mathbf{A} \otimes \mathbf{B} \equiv \begin{bmatrix} a_{1,1}\mathbf{B} & \dots & a_{1,n}\mathbf{B} \\ \vdots & \ddots & \vdots \\ a_{m,1}\mathbf{B} & \dots & a_{m,n}\mathbf{B} \end{bmatrix}$$

Theorem E.4. The Kronecker product has the following properties:

1. The rank of the Kronecker product of two matrices is equal to the product of their individual ranks, i.e.,

$$\text{rank}(\mathbf{A} \otimes \mathbf{B}) = \text{rank}(\mathbf{A})\text{rank}(\mathbf{B})$$

2. Furthermore,

$$(\mathbf{A}_1 \mathbf{A}_2 \dots \mathbf{A}_n) \otimes (\mathbf{B}_1 \mathbf{B}_2 \dots \mathbf{B}_n) = (\mathbf{A}_1 \otimes \mathbf{B}_1)(\mathbf{A}_2 \otimes \mathbf{B}_2) \dots (\mathbf{A}_n \otimes \mathbf{B}_n)$$

Proofs of these properties can be found in Horn and Johnson (1991).

Definition E.5 (Khatra-Rao product). *The Khatra-Rao product of two matrices $\mathbf{A} \in \mathbb{R}^{n \times mn}$, $\mathbf{B} \in \mathbb{R}^{m \times mn}$ is defined as the matrix that contains column-wise Kronecker products. Formally,*

$$\mathbf{A} \odot \mathbf{B} = \begin{bmatrix} \mathbf{A}_1 \otimes \mathbf{B}_1 & \mathbf{A}_2 \otimes \mathbf{B}_2 & \dots & \mathbf{A}_{mn} \otimes \mathbf{B}_{mn} \end{bmatrix}$$

where $\mathbf{A}_i, \mathbf{B}_i$ denote the i – th column of $\mathbf{A}$ and $\mathbf{B}$ respectively.

Definition E.6 (Hadamard Product). *Let $\mathbf{A} \in \mathbb{R}^{n \times m}$ and $\mathbf{B} \in \mathbb{R}^{n \times m}$; then their Hadamard product is defined as the element-wise product between the two matrices. Formally,*

$$(\mathbf{A} \circ \mathbf{B})_{i,j} = (\mathbf{A})_{i,j} (\mathbf{B})_{i,j}$$

Theorem E.7 (Mixed products). *The products defined before are connected in the following ways:*

1. *The Khatra-Rao and the Kronecker product satisfy*

$$(\mathbf{AC}) \odot (\mathbf{BD}) = (\mathbf{A} \otimes \mathbf{B})(\mathbf{C} \odot \mathbf{D})$$

2. *The Khatra-Rao and the Hadamard product satisfy*

$$(\mathbf{A} \odot \mathbf{B}) \circ (\mathbf{C} \odot \mathbf{D}) = (\mathbf{A} \circ \mathbf{C}) \odot (\mathbf{B} \circ \mathbf{D})$$

The above two properties can be found in Liu and Trenkler (2008) and Horn and Johnson (1991).

E.1.3 Linear Systems

We state below the well-known theorem of Rouché-Kronecker-Capelli, about the existence of solutions of Linear systems.

Theorem E.8 (Rouché-Kronecker-Capelli). *Let $\mathbf{Ax} = \mathbf{b}$ be a linear system of equations, with $\mathbf{A} \in \mathbb{R}^{m \times n}$, $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{b} \in \mathbb{R}^m$. Then this system has*

1. No solution: *if $\text{rank}(\mathbf{A}) \neq \text{rank}([\mathbf{A}|\mathbf{b}])$.*
2. Unique solution: *if $\text{rank}(\mathbf{A}) = \text{rank}([\mathbf{A}|\mathbf{b}]) = n$.*
3. Infinite solutions: *if $\text{rank}(\mathbf{A}) = \text{rank}([\mathbf{A}|\mathbf{b}]) < n$.*

This theorem essentially states that if $\mathbf{b}$ is in the column space of $\mathbf{A}$, then the system has a solution.

Lemma E.9. Let $\mathbf{C} \in \mathbb{R}^{n \times nm}$, $\mathbf{X} \in \mathbb{R}^{nm \times nm}$ a diagonal matrix, $\mathbf{B} \in \mathbb{R}^{nm \times m}$ and $\mathbf{W}^* \in \mathbb{R}^{n \times m}$ then the solution of

$$\mathbf{CXB} = \mathbf{W}^*$$

with respect to the elements of the matrix $\mathbf{X}$ can be recasted as

$$(\mathbf{C} \odot \mathbf{B}^T) \text{vecd}(\mathbf{X}) = \text{vec}(\mathbf{W}^*)$$

where we use vecd to denote the vector consisting of only the diagonal elements of the diagonal matrix $\mathbf{X}$ and vec is the row-major vectorization of $\mathbf{W}^*$.

Proof. We want to show that there exists a choice of the elements of $\mathbf{X}$ such that

$$\mathbf{CXB} = \mathbf{W}^*$$

where $\mathbf{X}$ is a diagonal matrix. Equivalently, we can rewrite the above equation as

$$\begin{bmatrix} \sum_i c_{1,i} x_i b_{i,1} & \sum_i c_{1,i} x_i b_{i,2} & \dots & \sum_i c_{1,i} x_i b_{i,m} \\ \sum_i c_{2,i} x_i b_{i,1} & \sum_i c_{2,i} x_i b_{i,2} & \dots & \sum_i c_{2,i} x_i b_{i,m} \\ \vdots & \ddots & \dots & \vdots \\ \sum_i c_{n,i} x_i b_{i,1} & \sum_i c_{n,i} x_i b_{i,2} & \dots & \sum_i c_{n,i} x_i b_{i,m} \end{bmatrix} = \begin{bmatrix} w_{1,1}^* & w_{1,2}^* & \dots & w_{1,m}^* \\ w_{2,1}^* & w_{2,2}^* & \dots & w_{2,m}^* \\ \vdots & \ddots & \dots & \vdots \\ w_{n,1}^* & w_{n,2}^* & \dots & w_{n,m}^* \end{bmatrix}$$

for some choice of the elements of $\mathbf{X}$.

We rewrite the above matrix equation as a linear system and we have

$$\left\{ \begin{array}{l} \sum_i c_{1,i} x_i b_{i,1} = w_{1,1}^* \\ \sum_i c_{1,i} x_i b_{i,2} = w_{1,2}^* \\ \vdots \\ \sum_i c_{1,i} x_i b_{i,m} = w_{1,m}^* \\ \vdots \\ \sum_i c_{n,i} x_i b_{i,1} = w_{n,1}^* \\ \sum_i c_{n,i} x_i b_{i,2} = w_{n,2}^* \\ \vdots \\ \sum_i c_{n,i} x_i b_{i,m} = w_{n,m}^* \end{array} \right.$$

Finally, we rewrite the linear system in a matrix form and we have

$$\begin{bmatrix} c_{1,1}b_{1,1} & c_{1,2}b_{2,1} & \dots & c_{1,nm}b_{nm,1} \\ c_{1,1}b_{1,2} & c_{1,2}b_{2,2} & \dots & c_{1,nm}b_{nm,2} \\ \vdots & \ddots & \dots & \vdots \\ c_{1,1}b_{1,m} & c_{1,2}b_{2,m} & \dots & c_{1,nm}b_{nm,m} \\ \vdots & \vdots & \vdots & \vdots \\ c_{n,1}b_{1,1} & c_{n,2}b_{2,1} & \dots & c_{n,nm}b_{nm,1} \\ c_{n,1}b_{1,2} & c_{n,2}b_{2,2} & \dots & c_{n,nm}b_{nm,2} \\ \vdots & \ddots & \dots & \vdots \\ c_{n,1}b_{1,m} & c_{n,2}b_{2,m} & \dots & c_{n,nm}b_{nm,m} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \\ \vdots \\ x_{nm-m+1} \\ x_{nm-m+2} \\ \vdots \\ x_{nm} \end{bmatrix} = \begin{bmatrix} w_{1,1}^* \\ w_{1,2}^* \\ \vdots \\ w_{1,m}^* \\ \vdots \\ w_{n,1}^* \\ w_{n,2}^* \\ \vdots \\ w_{n,m}^* \end{bmatrix}$$

The matrix appearing in this equation is the so-called Khatri-Rao product (or column-wise Kronecker product) of the $\mathbf{C}, \mathbf{B}^T$. □

E.1.4 Results on random matrices

Khatri-Rao product and rank.¹ The full rankness of the Khatri-Rao for random matrices has been proven in Jiang et al. (2001), over the set of complex numbers. This however

¹We would like to note that while our initial proof for this result was complicated, during the appendix deadline we were able to construct a simple one.

doesn't instantly imply that the result holds for the reals too. We thus prove it here for completeness. We will first introduce a lemma, that has been proven in (Caron and Traynor, 2005) and shows that if a polynomial is not the zero polynomial then the set that results to zero value of the polynomial is of measure zero.

Lemma E.10. *Let $p(\mathbf{x})$ be a polynomial of degree d , $\mathbf{x} \in \mathbb{R}^n$. If p is not the zero polynomial, then the set*

$$\mathcal{S} := \{\mathbf{x} \in \mathbb{R}^n | p(\mathbf{x}) = 0\}$$

*is of measure zero.*²

Lemma E.11. *Let $\mathbf{A}, \mathbf{B}$ be matrices whose elements are drawn independently from some continuous distributions, with $\mathbf{A} \in \mathbb{R}^{n \times nm}$, $\mathbf{B} \in \mathbb{R}^{m \times nm}$. Then the matrix forming their Khatri-Rao product is full rank with probability 1.*

Proof. We rewrite here the Khatri-Rao product of two matrices for clarity

$$\mathbf{A} \odot \mathbf{B} = \begin{bmatrix} a_{1,1}\mathbf{B}_1 & \dots & a_{1,nm}\mathbf{B}_{nm} \\ \vdots & \dots & \vdots \\ a_{n,1}\mathbf{B}_1 & \dots & a_{n,nm}\mathbf{B}_{nm} \end{bmatrix}$$

Using Theorem E.10 we have that it is sufficient to find an assignment of the values, such that the determinant of the matrix is not the zero polynomial. One such assignment is the following

$$\mathbf{A} = \begin{bmatrix} \overbrace{1 \dots 1}^{m \text{ columns}} & \overbrace{0 \dots 0}^{m \text{ columns}} & \overbrace{0 \dots 0}^{m \text{ columns}} \\ 0 \dots 0 & \overbrace{1 \dots 1}^{m \text{ columns}} & 0 \dots 0 \dots \\ 0 \dots 0 & 0 \dots 0 & \overbrace{1 \dots 1}^{m \text{ columns}} \dots \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & \dots \\ 0 \dots 0 & 0 \dots 0 & 0 & \dots & 0 \dots 0 \dots \end{bmatrix}$$

And for the choice of $\mathbf{B}$ we have

$$\mathbf{B} = [\mathbf{I}_m | \dots | \mathbf{I}_m]$$

Then the Khatri-Rao of $\mathbf{A} \odot \mathbf{B}$ is equal to the identity matrix and so its determinant is not identical to zero and this completes the proof. $\square$

²Specifically Lebesgue measure zero.

Remark E.12. *If we have the Khatri-Rao product of four or three random matrices, i.e., $(\mathbf{AB}) \odot (\mathbf{CD})$ or $\mathbf{A} \odot (\mathbf{CD})$, where $\mathbf{A} \in \mathbb{R}^{n \times nm}$, $\mathbf{B} \in \mathbb{R}^{nm \times nm}$, $\mathbf{C} \in \mathbb{R}^{m \times nm}$ and $\mathbf{D} \in \mathbb{R}^{nm \times nm}$ then we can set $\mathbf{D}, \mathbf{B}$ to be the identity and we just go back to the Khatri-Rao of two matrices.*

E.2 Width Overparameterization

Before we proceed with the theorems and the proofs of this section we remind the following key concepts & assumptions used:

- When referring to a randomly initialized neural network with BatchNorm it means that all parameters (weights, mean and variance) are randomly initialized from an arbitrary continuous and bounded distribution except for the scaling and shifting parameters which we can control.
- Unless stated otherwise $\|\mathbf{x}\|_2 \leq 1$.
- We will say that a network has rank r if the rank of the matrix before the activation function is r .

We start by showing that almost surely any two layer random ReLU network with BatchNorm, of width d^2 can reconstruct any one layer ReLU network with BatchNorm and d width, by tuning the normalization scaling and shifting parameters.

Proposition E.13. *Any layer of a linear network with BatchNorm, i.e.,*

$$f(\mathbf{x}) = \Gamma \frac{\mathbf{W}\mathbf{x} - \boldsymbol{\mu}}{s} + \beta,$$

can be equivalently expressed as

$$f(\mathbf{x}) = \Gamma' \mathbf{W}\mathbf{x} + \beta'$$

Proof. The proof follows trivially by setting $\Gamma' = \Gamma/s$ and $\beta' = \beta - \Gamma'\boldsymbol{\mu}$ □

Lemma E.14. *Consider a randomly initialized two layer ReLU network f with Batch Normalization and width d^2 ; then by appropriately choosing the scaling and shifting parameters of the BatchNorm, the network can be functionally equivalent with any one layer network h of the same architecture and width d with probability 1.*

Proof. Using Theorem E.13 we can write the one layer ReLU network with BatchNorm as

$$h(\mathbf{x}) = \sigma(\mathbf{\Gamma}^* \mathbf{W}^* \mathbf{x} + \beta^*)$$

where $\mathbf{\Gamma}^*, \mathbf{W}^* \in \mathbb{R}^{d \times d}$ and $\beta^* \in \mathbb{R}^d$.

The output of a two layer neural network with two layer of Batch Normalization, can be written as

$$f(\mathbf{x}) = \sigma \left(\mathbf{\Gamma}' \frac{\mathbf{V} \sigma \left(\mathbf{\Gamma} \frac{\mathbf{W} \mathbf{x} - \boldsymbol{\mu}}{s} + \beta \right) - \boldsymbol{\mu}'}{s'} + \beta' \right)$$

where $\mathbf{\Gamma} \in \mathbb{R}^{d^2 \times d^2}$, $\mathbf{\Gamma}' \in \mathbb{R}^{d \times d}$ and they are diagonal matrices, $\beta \in \mathbb{R}^{d^2}$ and $\beta' \in \mathbb{R}^d$. Also $\mathbf{W} \in \mathbb{R}^{d^2 \times d}$ and $\mathbf{V} \in \mathbb{R}^{d \times d^2}$.

The output of the i -th neuron in first layer before the ReLU activation is a vector with values

$$f_i^1(\mathbf{x}) = \frac{1}{s} \gamma_i \langle \mathbf{W}_i, \mathbf{x} \rangle - \frac{1}{s} \gamma_i \boldsymbol{\mu}_i + \beta_i$$

Recall that we assume $\|\mathbf{x}\|_2 \leq 1$, and that the inicializations are from a bounded distribution, say $\|\mathbf{W}\| \leq C^3$. Further, let $c = \max\{|\boldsymbol{\mu}_i|\}$ and $c' = |1/s|$. Then,

$$\begin{aligned} -\frac{1}{s} \gamma_i \langle \mathbf{W}_i, \mathbf{x} \rangle + \frac{1}{s} \gamma_i \boldsymbol{\mu}_i &\leq \left| \frac{1}{s} \gamma_i \langle \mathbf{W}_i, \mathbf{x} \rangle - \frac{1}{s} \gamma_i \boldsymbol{\mu}_i \right| \\ &\leq |\gamma_i/s| |\langle \mathbf{W}_i, \mathbf{x} \rangle - \boldsymbol{\mu}_i| \\ &\leq c' |\gamma_i| (|\langle \mathbf{W}_i, \mathbf{x} \rangle| + |\boldsymbol{\mu}_i|) \\ &\leq c' |\gamma_i| (\|\mathbf{W}_i\|_2 \|\mathbf{x}\|_2 + |\boldsymbol{\mu}_i|) \\ &\leq c' (C + c) |\gamma_i|. \end{aligned}$$

Hence, if $\beta_i \geq C' |\gamma_i|$ for $C' = c'(c + C)$, then $\sigma(f_i^1(\mathbf{x})) = f_i^1(\mathbf{x})$ because $f_i^1(\mathbf{x}) \geq 0$. Then, the output of the Neural Network is

$$f(\mathbf{x}) = \sigma \left(\frac{1}{ss'} \mathbf{\Gamma}' \mathbf{V} \mathbf{\Gamma} \mathbf{W} \mathbf{x} - \frac{1}{ss'} \mathbf{\Gamma}' \mathbf{V} \mathbf{\Gamma} \boldsymbol{\mu} + \frac{1}{s'} \mathbf{\Gamma}' \mathbf{V} \beta - \frac{1}{s'} \mathbf{\Gamma}' \boldsymbol{\mu}' + \beta' \right)$$

Setting $\mathbf{\Gamma}' := ss' \mathbf{\Gamma}^*$ and $\beta' := -\frac{1}{s'} \mathbf{\Gamma}' \mathbf{V} \beta + \frac{1}{ss'} \mathbf{\Gamma}' \mathbf{V} \mathbf{\Gamma} \boldsymbol{\mu} + \frac{1}{s'} \mathbf{\Gamma}' \boldsymbol{\mu}' + \beta^*$, we get

$$f(\mathbf{x}) = \sigma(\mathbf{\Gamma}^* \mathbf{V} \mathbf{\Gamma} \mathbf{W} \mathbf{x} + \beta^*)$$

In order to prove that this Neural Network can exactly express any layer $h(\mathbf{x}) = \sigma(\mathbf{\Gamma}^* \mathbf{W}^* \mathbf{x} +$

³We use here the Forbenius norm.

$\beta^*)$ it is sufficient to show that

$$\mathbf{V}\mathbf{T}\mathbf{W} = \mathbf{W}^*$$

By Theorem E.9, combined with Theorem E.8 it is sufficient to prove that the Khatri-Rao product of $\mathbf{V}, \mathbf{W}^T$ is full rank, then the linear system will have a unique solution. Finally, using Theorem E.11 concludes the proof. $\square$

Remark E.15. *Since we showed in the proof of Theorem E.14, how we can incorporate the Batch Normalization parameters of the targeted network in our analysis, we are going to ignore them from now on, for simplicity of exposition.*

Having established this result, we can show that if the targeted neural network is in fact low rank, by doubling the depth, we just need width that scales with the rank. When we say that the neural network is low rank, this can imply two facts one the scaling parameter of the BatchNorm has some zeros or that the weight matrix is low rank. In the first case, it follows trivially from the above proof that we will just have to focus on the non-zero rows since we can match the zeros one by choosing the corresponding scaling parameters to be zero. We treat the second case below.

Corollary E.16. *Consider a randomly initialized four layer ReLU network with Batch Normalization and width dr ; then by appropriately choosing the scaling and shifting parameters of the BatchNorm, the network can be functionally equivalent with any one layer network of the same architecture, width d and weight matrix of $r < d$ with probability 1 .*

Proof. Since the weight matrix $\mathbf{W}^*$ has rank $r < d$, we can use SVD decomposition to write this matrix as two low rank matrices, i.e.,

$$\mathbf{W}^* = \mathbf{A}^* \mathbf{B}^*$$

where $\mathbf{A}^* \in \mathbb{R}^{d \times r}$ and $\mathbf{B}^* \in \mathbb{R}^{r \times d}$. We will only show how to reconstruct this matrix, since the rest of the proof follows as that of Theorem E.14. We will use the first two layers to reconstruct $\mathbf{B}^*$ and the last two layers to reconstruct $\mathbf{A}^*$. More specifically, as we did in the proof of Theorem E.14, we first linearize the relu and then we want to show that the following linear system has a solution

$$\mathbf{V}_1 \mathbf{\Gamma}_1 \mathbf{W}_1 = \mathbf{B}^*$$

where $\mathbf{V}_1 \in \mathbb{R}^{r \times dr}$, $\mathbf{\Gamma}_1 \in \mathbb{R}^{dr \times dr}$ and $\mathbf{W}_1 \in \mathbb{R}^{dr \times d}$. This system has a solution as we showed in Theorem E.14. We now consider as $\mathbf{x}$ the output of these two layers which will be $\mathbf{B}^* \mathbf{x}$ after the reconstruction and we use the next two layers to reconstruct $\mathbf{A}^*$. $\square$

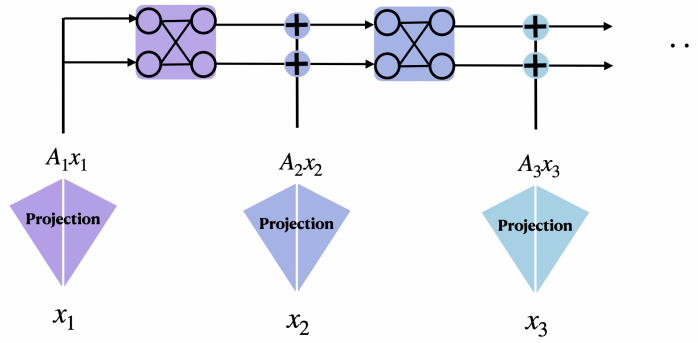


Figure E.1: A visualization of the skip connections. $\mathbf{x}_i$ is the i -th partition of the input $\mathbf{x}$.

Remark E.17. We note here we can also treat $\Gamma^* \mathbf{W}^*$ as one matrix and scale the width w.r.t. the rank of this matrix.

Theorem E.18 (Restatement of Theorem 9.5). Let $g(\mathbf{x}) = \mathbf{W}_l^* \sigma(\mathbf{W}_{l-1}^* \sigma(\dots \sigma(\mathbf{W}_1^* \mathbf{x})))$ be any ReLU network of rank r , depth l , where $\mathbf{x} \in \mathbb{R}^d$ and $\mathbf{W}_i^* \in \mathbb{R}^{d \times d}$ with $\|\mathbf{W}_i^*\| \leq 1$ for all $i = 1, \dots, l$. Consider a randomly initialized ReLU network of depth $2l$ with Batch Normalization:

$$f(\mathbf{x}) = \Gamma_{2l} \mathbf{W}_{2l} \sigma(\Gamma_{2l-1} \mathbf{W}_{2l-1} \sigma(\dots \sigma(\Gamma_1 \mathbf{W}_1 \mathbf{x} + \beta_1)) + \beta_{2l-1}) + \beta_{2l}$$

where $\Gamma_i \in \mathbb{R}^{d^2 \times d^2}$, $\mathbf{W}_i \in \mathbb{R}^{d \times d^2}$ for all odd $i \in [2l]$ and $\Gamma_i \in \mathbb{R}^{d \times d}$, $\mathbf{W}_i \in \mathbb{R}^{d^2 \times d}$ for all even $i \in [2l]$. Then one can compute the Batch Normalization parameters $\Gamma_1 \dots \Gamma_{2l}$ and $\beta_1 \dots \beta_{2l}$, so that with probability 1 the two networks are equivalent $f \equiv g$, i.e., $f(\mathbf{x}) = g(\mathbf{x})$ for all $\mathbf{x} \in \mathbb{R}^d$ with $\|\mathbf{x}\| \leq 1$.

Proof. To prove this theorem, we apply Theorem E.14 layer-wise for every layer of g , to get corresponding two layers of f . $\square$

Remark E.19. The size of the network can be the same throughout all layers, by zero padding the matrices, without changing the result.

E.3 Depth Overparameterization

In this section we will show how the combination of a form of skip connections and depth can lead to the same results, with a reduced width at each layer. In Fig. E.1 we have an illustration of the architecture used.

Lemma E.20. Let $\mathbf{A} \in \mathbb{R}^{n \times nm}$, $\mathbf{C} \in \mathbb{R}^{m \times nm}$ be random matrices, $\mathbf{D} \in \mathbb{R}^{nm \times nm}$ a diagonal fixed matrix with non-zero elements and w^* a fixed vector which is not the zero vector. Define $\mathbf{B} = (\mathbf{AD}) \odot \mathbf{C}$, then

$$\mathbb{P}(\langle \mathbf{B}_i^{-1}, w^* \rangle = 0) = 0$$

where $\mathbf{B}_i^{-1}$ is the i -th row of the inverse of $\mathbf{B}$.

Proof. Without loss of generality, we will focus on the first row of $\mathbf{B}^{-1}$, $\mathbf{B}_1^{-1}$ and we will drop the index 1. We know that the elements of this row are $(\mathbf{B}^{-1})_i = \frac{1}{\text{Det}(\mathbf{B})}(-1)^{1+i}\text{Det}_{-i,-1}$, where $\text{Det}_{-i,-1}$ is the determinant of matrix $\mathbf{B}$ with the i -th row and the first column deleted. So

$$\langle \mathbf{B}^{-1}, w^* \rangle = \sum_{i=1}^{nm} (-1)^{1+i} w_i^* \text{Det}_{-i,-1} \quad (\text{E.1})$$

where we skipped the term $\frac{1}{\text{Det}(\mathbf{B})}$ because it doesn't affect whether the inner product is zero or not. Notice that Eq. (E.1) is actually the determinant of $\mathbf{B}$ if we substitute the first column with the vector w^* . Our matrix is now

$$\mathbf{B}' = \begin{bmatrix} w_1^* & a_{1,2}d_2c_{1,2} & \dots & a_{1,nm}d_{nm}c_{1,nm} \\ w_2^* & a_{1,2}d_2c_{2,2} & \dots & a_{1,nm}d_{nm}c_{2,nm} \\ \vdots & \vdots & \ddots & \vdots \\ w_m^* & a_{1,2}d_2c_{m,2} & \dots & a_{1,nm}d_{nm}c_{m,nm} \\ \vdots & \vdots & \ddots & \vdots \\ w_{nm-m+1}^* & a_{n,2}d_2c_{1,2} & \dots & a_{n,nm}d_{nm}c_{1,nm} \\ \vdots & \vdots & \ddots & \vdots \\ w_{nm}^* & a_{n,2}d_2c_{m,2} & \dots & a_{n,nm}d_{nm}c_{m,nm} \end{bmatrix}$$

Since w^* is not a zero vector, there exist at least one non zero element of w^* . Again without loss of generality, assume that w_1^* is that element. Then, we assign to the matrices $\mathbf{A}$, $\mathbf{C}$ the same values as in the proof of Theorem 9.12. In that case, the matrix is now the diagonal matrix $\mathbf{D}$ with the first row substituted by the vector w^* . This matrix is full rank, since all columns and rows are linearly independent. $\square$

Theorem E.21 (Restatement of Theorem 9.7). Consider the following neural network

$$g(\mathbf{x}) = \mathbf{W}_l^* \sigma(\dots \sigma(\mathbf{W}_1^* \mathbf{x}))$$

where $\mathbf{W}_i^* \in \mathbb{R}^{d \times d}$ with $\|\mathbf{W}_i^*\| \leq 1$ for all $i = 1, \dots, l$.

Let the $j(\lceil \frac{d}{k} \rceil + 1)$ -th layer of the neural network f be

$$f^j(\mathbf{z}) = \sigma(\mathbf{W}_{\lceil \frac{d}{k} \rceil + 1}^j L_{\lceil \frac{d}{k} \rceil}^j) \text{ for } j = 1, \dots, l,$$

where, $\mathbf{z} \in \mathbb{R}^d$,

$$\mathbf{W}_{\lceil \frac{d}{k} \rceil + 1}^j \in \mathbb{R}^{d \times dk}, \text{ for } j = 1, \dots, l$$

$$L_1^j = \mathbf{\Gamma}_1^j \mathbf{W}_1^j \mathbf{z}_1^j + \beta_1^j,$$

$$L_i^j = \mathbf{\Gamma}_i^j \mathbf{W}_i^j [\sigma(L_{i-1}^j) + \mathbf{A}_i^j \mathbf{z}_i^j] + \beta_i^j \text{ for } i = 2, \dots, \lceil \frac{d}{k} \rceil$$

$$\mathbf{W}_i^j, \mathbf{\Gamma}_i^j \in \mathbb{R}^{dk \times dk}, \mathbf{A}_i^j \in \mathbb{R}^{dk \times k}, \text{ and } \beta_i^j \in \mathbb{R}^{dk} \text{ for all } i = 1, \dots, \lceil \frac{d}{k} \rceil, j = 1, \dots, l.$$

Here the matrices $\mathbf{W}_i^j, \forall i, j$ are randomly initialized and then frozen. Then, with probability 1, one can compute the BatchNorm parameters so that the two networks are equivalent, meaning

$$g(\mathbf{x}) = f^l(f^{l-1}(\dots f^1(\mathbf{x}))), \forall \|\mathbf{x}\| \leq 1$$

Here, the parameter k is tunable and can be any integer in $[d]$.

Remark E.22. Before we proceed with the proof of this theorem we would like to highlight that for the last layer, we would need not to include the ReLU in the constructed neural network.

Proof. We start by giving some intuition on how to use the skip connections in order to employ deeper but thinner networks. The idea is to partition the input, pass a different piece of the input in each individual layer and try to exactly reconstruct the parts of the target matrix $\mathbf{W}^*$ that correspond to the specific piece. Let k be the size of each partition and so in total we need to have $\lceil \frac{d}{k} \rceil$ layers. More specifically, notice that

$$\begin{aligned} \mathbf{W}^* \mathbf{x} &= \begin{bmatrix} \langle \mathbf{W}_1^*, \mathbf{x} \rangle \\ \vdots \\ \langle \mathbf{W}_d^*, \mathbf{x} \rangle \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^d w_{1,i}^* \mathbf{x}_i \\ \vdots \\ \sum_{i=1}^d w_{d,i}^* \mathbf{x}_i \end{bmatrix} \\ &= \begin{bmatrix} \sum_{i=1}^k w_{1,i}^* \mathbf{x}_i + \sum_{i=k+1}^{2k} w_{1,i}^* \mathbf{x}_i + \dots + \sum_{i=(\lceil \frac{d}{k} \rceil - 1)k+1}^d w_{1,i}^* \mathbf{x}_i \\ \vdots \\ \sum_{i=1}^k w_{d,i}^* \mathbf{x}_i + \sum_{i=k+1}^{2k} w_{d,i}^* \mathbf{x}_i + \dots + \sum_{i=(\lceil \frac{d}{k} \rceil - 1)k+1}^d w_{d,i}^* \mathbf{x}_i \end{bmatrix} \end{aligned}$$

$$\begin{aligned}
&= \begin{bmatrix} \sum_{i=1}^k w_{1,i}^* \mathbf{x}_i \\ \vdots \\ \sum_{i=1}^k w_{d,i}^* \mathbf{x}_i \end{bmatrix} + \begin{bmatrix} \sum_{i=k+1}^{2k} w_{1,i}^* \mathbf{x}_i \\ \vdots \\ \sum_{i=k+1}^{2k} w_{d,i}^* \mathbf{x}_i \end{bmatrix} + \dots + \begin{bmatrix} \sum_{i=(\lceil \frac{d}{k} \rceil - 1)k+1}^d w_{1,i}^* \mathbf{x}_i \\ \vdots \\ \sum_{i=(\lceil \frac{d}{k} \rceil - 1)k+1}^d w_{d,i}^* \mathbf{x}_i \end{bmatrix} \\
&= \mathbf{W}^* \mathbf{S}_1^T \mathbf{S}_1 \mathbf{x} + \mathbf{W}^* \mathbf{S}_2^T \mathbf{S}_2 \mathbf{x} + \dots + \mathbf{W}^* \mathbf{S}_{\lceil \frac{d}{k} \rceil}^T \mathbf{S}_{\lceil \frac{d}{k} \rceil} \mathbf{x} \quad (\text{E.2})
\end{aligned}$$

where $\mathbf{S}_i \in \mathbb{R}^{k \times d}$ is a sub-selection matrix and specifically

$$\mathbf{S}_i = [\mathbf{0} \mid \mathbf{I}_k \mid \mathbf{0}]$$

where the identity is in the place of $[(i-1)k+1, ik]$ columns.

Having this in mind, we will use each one of the layers with the skip connections to approximate a specific part of the input, which is then projected to a dk -dimensional vector through the matrix $\mathbf{A}_i$.

$$f(\mathbf{x}) = \mathbf{W}_{\lceil \frac{d}{k} \rceil + 1} L_{\lceil \frac{d}{k} \rceil}$$

where $L_1 = \Gamma_1 \mathbf{W}_1 \mathbf{x}_1 + \beta_1$ and $L_i = \Gamma_i \mathbf{W}_i [\sigma(L_{i-1}) + \mathbf{A}_i \mathbf{x}_i] + \beta_i$ for $i = 2, \dots, \lceil \frac{d}{k} \rceil - 1$. Before we proceed further, let us note the dimensions of each matrix and vector. 1) $\mathbf{x}_i = \mathbf{S}_i \mathbf{x} \in \mathbb{R}^k$ 2) $\mathbf{A}_i \in \mathbb{R}^{dk \times k}$ is the matrix that projects the input to a higher dimension and it is randomly initialized 3) $\mathbf{W}_i \in \mathbb{R}^{dk \times dk}$ randomly initialized 4) $\Gamma_i \in \mathbb{R}^{dk \times dk}$, $\beta_i \in \mathbb{R}^{dk}$ the parameters that we control for all $i = 1, \dots, \lceil \frac{d}{k} \rceil$. The last linear layer is added to correct the dimensions and project everything back to $\mathbb{R}^d$; thus $\mathbf{W}_{\lceil \frac{d}{k} \rceil + 1} \in \mathbb{R}^{d \times dk}$.

Linearization. We will set the parameters β_i , $i = 1, \dots, \lceil \frac{d}{k} \rceil - 1$ to be such that all ReLUs are activated as we did in the proof of Theorem E.18, while by controlling $\beta_{\lceil \frac{d}{k} \rceil}$ we will cancel out any error created.

Segmentation. After the previous step has been completed, the output of the network becomes

$$f(\mathbf{x}) = \mathbf{W}_{\lceil \frac{d}{k} \rceil + 1} \left(\prod_{i=1}^{\lceil \frac{d}{k} \rceil} \Gamma_i \mathbf{W}_i \mathbf{A}_i \mathbf{x}_1 + \prod_{i=2}^{\lceil \frac{d}{k} \rceil} \Gamma_i \mathbf{W}_i \mathbf{A}_i \mathbf{x}_2 + \dots + \Gamma_{\lceil \frac{d}{k} \rceil} \mathbf{W}_{\lceil \frac{d}{k} \rceil} \mathbf{A}_{\lceil \frac{d}{k} \rceil} \mathbf{x}_{\lceil \frac{d}{k} \rceil} \right) \quad (\text{E.3})$$

We will now use each one of the terms in the summation to reconstruct each one of the terms in the last part of Eq. (E.2). Thus we will try to solve the system of linear systems. We will show that this is feasible, by using induction.

Base step. We start with the last equation which is the simplest and we have

$$\mathbf{W}_{\lceil \frac{d}{k} \rceil + 1} \mathbf{\Gamma}_{\lceil \frac{d}{k} \rceil} \mathbf{W}_{\lceil \frac{d}{k} \rceil} \mathbf{A}_{\lceil \frac{d}{k} \rceil} \mathbf{x}_{\lceil \frac{d}{k} \rceil} = \mathbf{W}_{\lceil \frac{d}{k} \rceil}^* \mathbf{x}_{\lceil \frac{d}{k} \rceil}$$

It is sufficient to show that

$$\mathbf{W}_{\lceil \frac{d}{k} \rceil + 1} \mathbf{\Gamma}_{\lceil \frac{d}{k} \rceil} \mathbf{W}_{\lceil \frac{d}{k} \rceil} \mathbf{A}_{\lceil \frac{d}{k} \rceil} = \mathbf{W}_{\lceil \frac{d}{k} \rceil}^*$$

where $\mathbf{W}_{\lceil \frac{d}{k} \rceil}^* = \mathbf{W}^* \mathbf{S}_{\lceil \frac{d}{k} \rceil}^T$. From Theorem 9.11 we know that the above system can be rewritten as

$$\mathbf{W}_{\lceil \frac{d}{k} \rceil + 1} \odot (\mathbf{A}_{\lceil \frac{d}{k} \rceil}^T \mathbf{W}_{\lceil \frac{d}{k} \rceil}^T) \text{vecd}(\mathbf{\Gamma}_{\lceil \frac{d}{k} \rceil}) = \text{vec}(\mathbf{W}_{\lceil \frac{d}{k} \rceil}^*)$$

which from Theorem E.12 is solvable with probability 1. By Theorem E.20 we also have that with probability 1 all the elements of $\mathbf{\Gamma}_{\lceil \frac{d}{k} \rceil}$ are non-zero.

Inductive step. Assume now that all components up to (and not included) the $i - th$ component have been exactly reconstructed and $\mathbf{\Gamma}_j, j = \lceil \frac{d}{k} \rceil, \dots, i - 1$ have non-zero elements with probability one.

Last step. We will now show that the $i - th$ component of Eq. (E.3) can exactly reconstruct the $i - th$ component of Eq. (E.2). We want

$$\mathbf{W}_{\lceil \frac{d}{k} \rceil + 1} \prod_{j=i}^{\lceil \frac{d}{k} \rceil} \mathbf{\Gamma}_j \mathbf{W}_j \mathbf{A}_i \mathbf{x}_i = \mathbf{W}_i^* \mathbf{x}_i$$

Again it is sufficient to show that

$$\mathbf{W}_{\lceil \frac{d}{k} \rceil + 1} \prod_{j=i}^{\lceil \frac{d}{k} \rceil} \mathbf{\Gamma}_j \mathbf{W}_j \mathbf{A}_i = \mathbf{W}_i^*$$

We can rewrite this system using Theorem 9.11 with respect to $\mathbf{\Gamma}_i$ as

$$\left[(\mathbf{W}_{\lceil \frac{d}{k} \rceil + 1} \prod_{j=i-1}^{\lceil \frac{d}{k} \rceil} \mathbf{\Gamma}_j \mathbf{W}_j) \odot (\mathbf{A}_i^T \mathbf{W}_i^T) \right] \text{vecd}(\mathbf{\Gamma}_i) = \text{vec}(\mathbf{W}_i^*)$$

We now need to show that the first matrix is full rank with probability one. From Theorem E.7 part 1 we can rewrite this matrix as

$$\begin{aligned}
& (\mathbf{W}_{\lceil \frac{d}{k} \rceil + 1} \prod_{j=i-1}^{\lceil \frac{d}{k} \rceil} \mathbf{\Gamma}_j \mathbf{W}_j) \odot (\mathbf{A}_i^T \mathbf{W}_i^T) \\
&= ((\mathbf{W}_{\lceil \frac{d}{k} \rceil + 1} \prod_{j=i-2}^{\lceil \frac{d}{k} \rceil} \mathbf{\Gamma}_j \mathbf{W}_j) \otimes \mathbf{A}_i^T) ((\mathbf{\Gamma}_{i-1} \mathbf{W}_{i-1}) \odot \mathbf{W}_i^T) \\
&= (\mathbf{W}_{\lceil \frac{d}{k} \rceil + 1} \otimes \mathbf{A}_i^T) \prod_{j=i-2}^{\lceil \frac{d}{k} \rceil} (\mathbf{\Gamma}_j \otimes \mathbf{I}_{dk}) (\mathbf{W}_j \otimes \mathbf{I}_{dk}) (\mathbf{\Gamma}_{i-1} \otimes \mathbf{I}_{dk}) (\mathbf{W}_{i-1} \odot \mathbf{W}_i^T)
\end{aligned}$$

where we applied for the last equation Theorem E.7 part 1 one more time and Theorem E.4 part 2 to split the Kronecker product to product of Kronecker products. We now want to argue about the rank of the matrix above. Notice that from Theorem E.4 part 1, all the intermediate Kronecker products result in matrices of rank $d^2 k^2$ which means that they are all full rank and thus they have an empty nullspace. Furthermore, from Theorem E.1 we have that

$$\begin{aligned}
& \text{Null}((\mathbf{W}_{\lceil \frac{d}{k} \rceil + 1} \otimes \mathbf{A}_i^T) \prod_{j=i-2}^{\lceil \frac{d}{k} \rceil} (\mathbf{\Gamma}_j \otimes \mathbf{I}_{dk}) (\mathbf{W}_j \otimes \mathbf{I}_{dk}) (\mathbf{\Gamma}_{i-1} \otimes \mathbf{I}_{dk})) \\
&= \text{Null}((\mathbf{W}_{\lceil \frac{d}{k} \rceil + 1} \otimes \mathbf{A}_i^T))
\end{aligned}$$

Thus, it is sufficient to show that the matrix $(\mathbf{W}_{\lceil \frac{d}{k} \rceil + 1} \otimes \mathbf{A}_i^T)(\mathbf{W}_{i-1} \odot \mathbf{W}_i^T)$ is full rank, meaning that the null space of the Kronecker product has no intersection with the column space of the Khatri-Rao product. Also, using again Theorem E.7 we have that

$$(\mathbf{W}_{\lceil \frac{d}{k} \rceil + 1} \otimes \mathbf{A}_i^T)(\mathbf{W}_{i-1} \odot \mathbf{W}_i^T) = (\mathbf{W}_{\lceil \frac{d}{k} \rceil + 1} \mathbf{W}_{i-1}) \odot (\mathbf{A}_i^T \mathbf{W}_i^T)$$

And by using Theorem E.12 we conclude that the matrix is full rank. Now, to also prove that $\mathbf{\Gamma}_i$ does not have any zero elements, we need to find an assignment of the elements of the matrices such that any $\text{vec}(\mathbf{W}_i^*)$ is not orthogonal to any of the rows of the inverse of the matrix $(\mathbf{W}_{\lceil \frac{d}{k} \rceil + 1} \prod_{j=i-1}^{\lceil \frac{d}{k} \rceil} \mathbf{\Gamma}_j \mathbf{W}_j) \odot (\mathbf{A}_i^T \mathbf{W}_i^T)$ with probability 1.

Notice that each of the elements of the inverse of a matrix $\mathbf{B}$ is

$B_{ij}^{-1} = \frac{1}{\text{Det}(\mathbf{B})} (-1)^{i+j} \text{Det}_{-j, -i}$, where $\text{Det}_{-j, -i}$ is the determinant of the matrix $\mathbf{B}$ if we delete the j -th row and the i -th column. Without loss of generality we will just pick the first

row of the inverse matrix and prove the result. Let

$$\mathbf{B} = ((\mathbf{W}_{\lceil \frac{d}{k} \rceil + 1} \prod_{j=i-1}^{\lceil \frac{d}{k} \rceil} \Gamma_j \mathbf{W}_j) \odot (\mathbf{A}_i^T \mathbf{W}_i^T))^{-1}$$

Thus, the first row of this matrix

$$\mathbf{B}_1 = [(-1)^2 \text{Det}_{-1, -1}, \dots, (-1)^{1+i} \text{Det}_{-i, -1}, \dots, (-1)^{1+dk} \text{Det}_{-dk, -1}].$$

We want to show that $\mathbb{P}(\langle \mathbf{B}_1, \text{vec}(\mathbf{W}_i^*) \rangle = 0) = 0$ for any fixed $\text{vec}(\mathbf{W}_i^*)$ that it's not of course identical to zero. From Theorem E.10 it is sufficient to find an assignment of the values of the matrices that gives non-zero result always. Let us choose all $\mathbf{W}_j = \mathbf{I}_{dk}$ for all $j = i, \dots, dk$. So, it is sufficient to find an assignment for the rest such that the first row of $((\mathbf{W}_{\lceil \frac{d}{k} \rceil + 1} \Gamma) \odot \mathbf{A}_i^T)^{-1}$ is not orthogonal to a fixed vector $\text{vec}(\mathbf{W}_i^*)$, where $\Gamma = \prod_{j=i-1}^{\lceil \frac{d}{k} \rceil} \Gamma_j$ and it is a diagonal matrix with non-zero elements. By using Theorem E.20 we conclude the proof. $\square$

Remark E.23. *Except for the total number of parameters, which we have already mentioned, we would like to pinpoint that the architecture in this case is slightly different from a fully connected neural network, since it had an extra linear layer, before the activation function.*

E.4 Sparse Matrix Inverse

We now show that the random weight matrices of Theorem E.18 can be randomly sparsified and retain the invertibility of the Khatri-Rao product with high probability. The intuition of this result lies in the fact that the invertibility of a sparsified matrix is mainly based on the sparsity patterns that are created.

To do so, we first define the notion of Boolean determinant⁴

Definition E.24. *The Boolean determinant of a $d \times d$ Boolean matrix $\mathbf{B}$ is recursively defined to be*

- if $d = 1$, $\text{Det}_{\text{Bool}}(\mathbf{B}) := \mathbf{B}_{0,0}$, that is, it is the only element of the matrix, and

⁴ For ease of notation we index the columns and rows of matrices starting from 0.

- if $d > 1$,

$$\text{Det}_{\text{Bool}}(\mathbf{B}) := \max_{i \in \{0, \dots, d\}} (\mathbf{B}_{i,0} \text{Det}_{\text{Bool}}(\mathbf{B}_{-(i,0)})),$$

where $\mathbf{B}_{i,j}$ is the element of $\mathbf{B}$ at i -th row and j -th column, and $\mathbf{B}_{-(i,j)}$ is the sub-matrix of $\mathbf{B}$ with i -th row and j -th column removed. Note that $\text{Det}_{\text{Bool}}(\mathbf{B}) \in \{0, 1\}$ for any Boolean matrix $\mathbf{B}$.

In what follows, we will prove some auxiliary lemmas in order to prove Theorem 9.9.

Lemma E.25. *Let $\mathbf{A}, \mathbf{B}$ be two $d^2 \times d^2$ Bernoulli matrices and $\mathbf{P} \in \mathbb{R}^{d \times d^2}$, $\mathbf{Q} \in \mathbb{R}^{d \times d^2}$ be two random matrices. Then $(\mathbf{P} \odot \mathbf{A}) \odot (\mathbf{Q} \odot \mathbf{B})$ is invertible if and only if $\text{Det}_{\text{Bool}}(\mathbf{A} \odot \mathbf{B}) = 1$.*

Proof. Let $\mathbf{U} = \mathbf{P} \odot \mathbf{A}$ and $\mathbf{V} = \mathbf{Q} \odot \mathbf{B}$. We will prove the statement by induction on square submatrices of $(\mathbf{P} \odot \mathbf{A}) \odot (\mathbf{Q} \odot \mathbf{B})$ and the corresponding square submatrices of $\mathbf{A} \odot \mathbf{B}$, that is, we prove that $\text{Det}(((\mathbf{P} \odot \mathbf{A}) \odot (\mathbf{Q} \odot \mathbf{B}))_{\mathcal{I}, \mathcal{J}}) \neq 0$ if and only if $\text{Det}_{\text{Bool}}((\mathbf{A} \odot \mathbf{B})_{\mathcal{I}, \mathcal{J}}) = 1$, where $\mathcal{I} \subset [d^2]$ is the set of selected rows and $\mathcal{J} \subset [d^2]$ is the set of selected columns.

For size 1 submatrices, that is, when $|\mathcal{I}| = 1$ and $|\mathcal{J}| = 1$, the submatrix of $(\mathbf{P} \odot \mathbf{A}) \odot (\mathbf{Q} \odot \mathbf{B})$ will contain just one element, which will be of the form $pqab$, where p, q, a and b are the elements of the $\mathbf{P}, \mathbf{Q}, \mathbf{A}$ and $\mathbf{B}$ respectively. The corresponding submatrix of $\mathbf{A} \odot \mathbf{B}$ will just contain ab , and hence with probability 1, $pqab \neq 0 \iff ab = 1$. This proves the base case.

Now, we let $\mathbf{U} = \mathbf{P} \odot \mathbf{A}$ and $\mathbf{V} = \mathbf{Q} \odot \mathbf{B}$ and we move on to the case for $|\mathcal{I}| = |\mathcal{J}| > 1$. Here, we want to show that with probability 1,

$$\text{Det}((\mathbf{U} \odot \mathbf{V})_{\mathcal{I}, \mathcal{J}}) = 0 \iff \text{Det}_{\text{Bool}}((\mathbf{A} \odot \mathbf{B})_{\mathcal{I}, \mathcal{J}}) = 0$$

We write the expression for determinant of $(\mathbf{U} \odot \mathbf{V})_{\mathcal{I}, \mathcal{J}}$ calculated along its first column. Without loss of generality, assume that $0 \in \mathcal{J}$. Then,

$$\text{Det}((\mathbf{U} \odot \mathbf{V})_{\mathcal{I}, \mathcal{J}}) = \sum_{i \in \mathcal{I}} \mathbf{A}_{\lfloor i/d \rfloor, 0} \mathbf{B}_{i \pmod{d}, 0} \mathbf{P}_{\lfloor i/d \rfloor, 0} \mathbf{Q}_{i \pmod{d}, 0} \text{Det}((\mathbf{U} \odot \mathbf{V})_{\mathcal{I} \setminus \{i\}, \mathcal{J} \setminus \{0\}}),$$

We can rewrite the summation above as

$$\text{Det}((\mathbf{U} \odot \mathbf{V})_{\mathcal{I}, \mathcal{J}}) = \sum_{j=0}^{d-1} \mathbf{A}_{j,0} \mathbf{P}_{j,0} \sum_{l \in \{l: jd+l \in \mathcal{I}\}} \mathbf{B}_{l,0} \mathbf{Q}_{l,0} \text{Det}((\mathbf{U} \odot \mathbf{V})_{\mathcal{I} \setminus \{jd+l\}, \mathcal{J} \setminus \{0\}}).$$

Note that $\mathbf{A}_{j,0} \mathbf{P}_{j,0}$ is independent of $\sum_{l \in \{l: jd+l \in \mathcal{I}\}} \mathbf{B}_{l,0} \mathbf{Q}_{l,0} \text{Det}((\mathbf{U} \odot \mathbf{V})_{\mathcal{I} \setminus \{jd+l\}, \mathcal{J} \setminus \{0\}})$, since $\mathbf{A}, \mathbf{P}, \mathbf{B}$, and $\mathbf{Q}$ are all independent of each other, and the columns of $\mathbf{U} \odot \mathbf{V}$ are also

independent. Since $\mathbf{P}$ and $\mathbf{Q}$ have elements coming from continuous density distribution, with probability 1, the above expression is 0 if and only if each term in the sum above is individually 0.⁵ That is, the expression above is 0 if and only if all the terms of the form $\text{Det}((\mathbf{U} \odot \mathbf{V})_{\mathcal{I} \setminus \{jd+l\}, \mathcal{J} \setminus \{0\}})$ are 0. Using the inductive hypothesis we get that

$$\text{Det}((\mathbf{U} \odot \mathbf{V})_{\mathcal{I} \setminus \{jd+l\}, \mathcal{J} \setminus \{0\}}) = 0 \iff \text{Det}_{\text{Bool}}((\mathbf{A} \odot \mathbf{B})_{\mathcal{I} \setminus \{jd+l\}, \mathcal{J} \setminus \{0\}}) = 0$$

. Hence with probability 1, $\text{Det}((\mathbf{U} \odot \mathbf{V})_{\mathcal{I}, \mathcal{J}}) = 0$ if and only if

$$\sum_{j=0}^{d-1} \mathbf{A}_{j,0} \sum_{l \in \{l: jd+l \in \mathcal{I}\}} \mathbf{B}_{l,0} \text{Det}_{\text{Bool}}((\mathbf{A} \odot \mathbf{B})_{\mathcal{I} \setminus \{jd+l\}, \mathcal{J} \setminus \{0\}}) = 0.$$

The above expression is 0 if and only if

$$\max_{j \in \{0, \dots, d-1\}, l \in \{l: jd+l \in \mathcal{I}\}} (\mathbf{A}_{j,0} \mathbf{B}_{l,0} \text{Det}_{\text{Bool}}((\mathbf{A} \odot \mathbf{B})_{\mathcal{I} \setminus \{jd+l\}, \mathcal{J} \setminus \{0\}})) = 0.$$

Noting that

$$\max_{j \in \{0, \dots, d-1\}, l \in \{l: jd+l \in \mathcal{I}\}} (\mathbf{A}_{j,0} \mathbf{B}_{l,0} \text{Det}_{\text{Bool}}((\mathbf{A} \odot \mathbf{B})_{\mathcal{I} \setminus \{jd+l\}, \mathcal{J} \setminus \{0\}})) = \text{Det}_{\text{Bool}}((\mathbf{A} \odot \mathbf{B})_{\mathcal{I}, \mathcal{J}})$$

concludes the proof for the induction step. $\square$

Lemma E.26. Let $\mathbf{u}_0, \dots, \mathbf{u}_{d-1}$ and $\mathbf{v}_0, \dots, \mathbf{v}_{d-1}$ be Bernoulli random variables with probability p of success; and let $\mathbf{x}_0, \dots, \mathbf{x}_{d^2-1}$ be Bernoulli random variables with probability q of success, all sampled i.i.d. , with $p \geq \sqrt{2qd}$. Then, for any subset $\mathcal{S} \subset \{0, \dots, d^2 - 1\}$,

$$\mathbb{P} \left(\sum_{i \in \mathcal{S}} \mathbf{u}_{\lfloor i/d \rfloor} \mathbf{v}_{i \pmod{d}} = 0 \right) \leq \mathbb{P} \left(\sum_{i \in \mathcal{S}} \mathbf{x}_i = 0 \right)$$

Proof. Consider any $\mathcal{S}$ and define the set $\mathcal{A}_l = \{j : j = i \pmod{d} \wedge l = \lfloor i/d \rfloor \text{ for some } i \in \mathcal{S}\}$ for $l = 0, \dots, d-1$. Then,

$$\sum_{i \in \mathcal{S}} \mathbf{u}_{\lfloor i/d \rfloor} \mathbf{v}_{i \pmod{d}} = \sum_{l=0}^{d-1} \mathbf{u}_l \sum_{j \in \mathcal{A}_l} \mathbf{v}_j$$

⁵If some of the term is not zero, then this is a polynomial with respect to the variables $(\mathbf{P}, \mathbf{Q})$ and it is zero in a set of measure zero.

Hence,

$$\begin{aligned} \mathbb{P}\left(\sum_{i \in \mathcal{S}} \mathbf{u}_{[i/d]} \mathbf{v}_i \pmod{d}\right) &= \mathbb{P}\left(\sum_{l=0}^{d-1} \mathbf{u}_l \sum_{j \in \mathcal{A}_l} \mathbf{v}_j = 0\right) \\ &= \mathbb{P}\left(\sum_{l=1}^{d-1} \mathbf{u}_l \sum_{j \in \mathcal{A}_l} \mathbf{v}_j = 0\right) \mathbb{P}\left(\mathbf{u}_0 \sum_{j \in \mathcal{A}_0} \mathbf{v}_j = 0 \middle| \sum_{l=1}^{d-1} \mathbf{u}_l \sum_{j \in \mathcal{A}_l} \mathbf{v}_j = 0\right). \end{aligned}$$

Looking at the term $\mathbb{P}\left(\mathbf{u}_0 \sum_{j \in \mathcal{A}_0} \mathbf{v}_j = 0 \middle| \sum_{l=1}^{d-1} \mathbf{u}_l \sum_{j \in \mathcal{A}_l} \mathbf{v}_j = 0\right)$, note that if $|\mathcal{A}_0 \cap \bigcup_{l=1}^{d-1} \mathcal{A}_l| = 0$, then $\mathbb{P}\left(\mathbf{u}_0 \sum_{j \in \mathcal{A}_0} \mathbf{v}_j = 0 \middle| \sum_{l=1}^{d-1} \mathbf{u}_l \sum_{j \in \mathcal{A}_l} \mathbf{v}_j = 0\right) = \mathbb{P}\left(\mathbf{u}_0 \sum_{j \in \mathcal{A}_0} \mathbf{v}_j = 0\right)$. Otherwise,

$$\mathbb{P}\left(\mathbf{u}_0 \sum_{j \in \mathcal{A}_0} \mathbf{v}_j = 0 \middle| \sum_{l=1}^{d-1} \mathbf{u}_l \sum_{j \in \mathcal{A}_l} \mathbf{v}_j = 0\right) \geq \mathbb{P}\left(\mathbf{u}_0 \sum_{j \in \mathcal{A}_0} \mathbf{v}_j = 0\right)$$

. In order to show this, assume without loss of generality, that $|\mathcal{A}_0 \cap \bigcup_{l=1}^{d-1} \mathcal{A}_l| = k$. This means that at least k elements that are contained in $\mathcal{A}_0$ is also contained in at least one of the other $\mathcal{A}_l$ sets, this immediately implies that at least k terms in the summation of $\sum_{j \in \mathcal{A}_0} \mathbf{v}_j$ are zero. Notice that

$$\begin{aligned} \mathbb{P}(\mathbf{u}_0 \sum_{j \in \mathcal{A}_0} \mathbf{v}_j = 0) &= \mathbb{P}(\mathbf{u}_0 = 0) + \mathbb{P}(\sum_{j \in \mathcal{A}_0} \mathbf{v}_j = 0) - \mathbb{P}(\{\mathbf{u}_0 = 0\} \cap \{\sum_{j \in \mathcal{A}_0} \mathbf{v}_j = 0\}) \\ &= \mathbb{P}(\mathbf{u}_0 = 0) + \mathbb{P}(\sum_{j \in \mathcal{A}_0} \mathbf{v}_j = 0) - \mathbb{P}(\{\mathbf{u}_0 = 0\}) \mathbb{P}(\{\sum_{j \in \mathcal{A}_0} \mathbf{v}_j = 0\}) \end{aligned}$$

Since $\mathbf{u}_0$ is independent of $\mathbf{v}_j$ s. It is obvious that if we know that at least one of the $\mathbf{v}_j$ s is zero, then the probability that the sum of them is zero is increased ($\mathbf{v}_j$ s are also independent to each other). Thus the more $\mathbf{v}_j$ s we know that are zero the more this function is increased. (We can write the last expression as $p + q^{|\mathcal{A}_0|-k} - pq^{|\mathcal{A}_0|-k}$ and it is easy to see that as k is increased the value of the expression is increased).

Thus, to maximize $\mathbb{P}\left(\sum_{i \in \mathcal{S}} \mathbf{u}_{[i/d]} \mathbf{v}_i \pmod{d}\right)$, we need to maximize the size of overlaps of the form $\mathcal{A}_0 \cap \bigcup_{l=1}^{d-1} \mathcal{A}_l$. To proceed now, let $a_l := |\mathcal{A}_l|$. Then,

$$\begin{aligned} \mathbb{P}\left(\sum_{i \in \mathcal{S}} \mathbf{u}_{[i/d]} \mathbf{v}_i \pmod{d}\right) &= \mathbb{P}\left(\sum_{l=0}^{d-1} \mathbf{u}_l \sum_{j \in \mathcal{A}_l} \mathbf{v}_j = 0\right) \\ &\leq \mathbb{P}\left(\sum_{l=0}^{d-1} \mathbf{u}_l \sum_{j=0}^{a_l-1} \mathbf{v}_j = 0\right). \end{aligned}$$

(This maximizes the overlap amongst $\mathcal{A}_l$'s)

Without loss of generality, assume that $a_0 \geq a_1 \geq \dots$. Then, let h be the smallest integer such

that $|\{i : a_i \geq h\}| \leq h$. Next, we claim that $|\mathcal{S}| =: n \leq 2dh$. To see how, let $\mathcal{T}_1 = \{i : a_i \geq h\}$ and $\mathcal{T}_2 = \{i : a_i < h\}$. Then, $n = \sum a_i = \sum_{i \in \mathcal{T}_1} a_i + \sum_{i \in \mathcal{T}_2} a_i \leq dh + dh = 2dh$, since $|\mathcal{T}_1| \leq h$, $a_i \leq h$ for a_i in $\mathcal{T}_2$, $a_i \leq d$ and $|\mathcal{T}_2| \leq d$.

Continuing on from the equation above,

$$\begin{aligned} \mathbb{P} \left(\sum_{i \in \mathcal{S}} \mathbf{u}_{\lfloor i/d \rfloor} \mathbf{v}_{i \pmod{d}} \right) &\leq \mathbb{P} \left(\sum_{l=0}^{d-1} \mathbf{u}_l \sum_{j=0}^{a_l-1} \mathbf{v}_j = 0 \right) \\ &\leq \mathbb{P} \left(\sum_{l=0}^{h-1} \mathbf{u}_l \sum_{j=0}^{h-1} \mathbf{v}_j = 0 \right). \end{aligned}$$

The sum above is 0 if and only if $\sum_{l=0}^{h-1} \mathbf{u}_l = 0$ or $\sum_{j=0}^{h-1} \mathbf{v}_j = 0$. The probability of that is $2(1-p)^h - (1-p)^{2h} = (1-p)^h(2 - (1-p)^h)$. On the other hand, $\mathbb{P}(\sum_{i \in \mathcal{S}} \mathbf{x}_i = 0) = (1-q)^n \geq (1-q)^{2dh}$. Hence, we need the following to be true:

$$\begin{aligned} (1-p)^h(2 - (1-p)^h) &\leq (1-q)^{2dh} \\ \text{or } (1-p)(2 - (1-p)^h)^{1/h} &\leq (1-q)^{2d} \end{aligned}$$

Note that $(2 - (1-p)^h)^{1/h} \leq 1+p$ and $(1-q)^{2d} \geq 1-2qd$. Hence, it is sufficient for the following to hold:

$$(1-p)(1+p) \leq 1-2qd.$$

From the above, we get that $p \geq \sqrt{2qd}$ suffices. $\square$

This lemma shows that if we replace a column of $\mathbf{A} \odot \mathbf{B}$ with i.i.d. Bernoulli random variables with probability of success q , then the probability that the new matrix has Boolean determinant zero is higher than that of the original one. To see how, just let $\mathcal{S}$ be the set of indices i for which $\text{Det}_{\text{Bool}}((\mathbf{A} \odot \mathbf{B})_{-(i,0)}) = 1$. Then, the LHS in the statement of this lemma is just the probability of $\text{Det}_{\text{Bool}}(\mathbf{A} \odot \mathbf{B}) = 0$ and RHS is the probability of the new matrix having Boolean determinant zero. Since the columns of $\mathbf{A} \odot \mathbf{B}$ are independent, we can keep replacing the columns one-by-one to get a matrix with all elements sampled i.i.d. from $\text{Bern}(q)$. Then, we can use the following lemma on the new matrix.

We now restate a theorem from Basak and Rudelson (2018). Though mentioned in the main part, we also state it here for completeness and clarity:

Theorem E.27 (Theorem 1.1 in (Basak and Rudelson, 2018)). *let $\mathbf{A} \in \mathbb{R}^{d \times d}$ be an a matrix with i.i.d. $\text{Ber}(p)$ entries. Then, there exist absolute constants $0 < c, \bar{c}, C < \infty$ such that for any*

$\epsilon > 0$, and ρ such that $n\rho \geq \log(1/\rho)$, we have

$$\mathbb{P}\left(\left\{(s_{\min}(\mathbf{A}) \leq \bar{c}\epsilon \exp(-C \frac{\log(1/\rho)}{\log(n\rho)} \sqrt{\frac{\rho}{n}})) \cap \Omega_o^c\right\}\right) \leq \epsilon + n^{-c}$$

where $s_{\min}$ is the minimum singular value of the matrix $\mathbf{A}$ and Ω_o is the event that any column or some row is identical to 0.

Remark E.28. It is easy to see that $n\rho \geq \log(1/\rho)$ is equivalent with $\rho \geq \log n/n - \log \log n$. we note here that we focus on the invertibility of matrix $\mathbf{A}$, that is the probability that $s_{\min}(\mathbf{A}) > 0$. As a result, let ϵ to be exponentially - actually arbitrarily - small and $\rho = \log n/n$; then the matrix $\mathbf{A}$ is non-singular with probability of the order $1/n^c$. We emphasize that the constant c is universal.

Theorem E.29 (Restatement of Theorem 9.9). For the setting of Theorem E.18, consider that the random matrices of each of the layers are sparsified with probability $p = \Theta(\sqrt{\log d/d})$, meaning that each of their elements is zero with probability $1 - p$. If the depth of the network is polynomial in the input, i.e., $l = \text{poly}(d)$ then the results of Theorem E.18 hold with probability at least $1 - 1/d$.

Proof. It is sufficient to prove that after sparsifying the matrices of the two first layers of our construction, i.e., $\mathbf{W}_1, \mathbf{W}_2$ with $p \geq \Theta(\sqrt{\log d/d})$, the probability that their Khatri-Rao product is no longer invertible is polynomially small in d . Let $\mathbf{M}_1, \mathbf{M}_2$ be $\text{Bern}(p)$ then from Theorem E.25 we have that it is sufficient that $\text{Det}_{\text{Bool}}(\mathbf{M}_1 \odot \mathbf{M}_2) \neq 0$. From Theorem E.26, we actually get that by substituting one column that has variables which are $\text{Bern}(q)$, with $p \geq \sqrt{2qd}$ results in matrix which has a higher probability of its Boolean determinant being zero. By applying this argument at each column we get a new matrix $\mathbf{M}$, which elements are i.i.d. $\text{Bern}(q)$.

We will now compute the value of q . Using Theorem 9.15, we get that every Bernoulli i.i.d. matrix $\text{Bern}(\rho)$, with $\rho = \log d^2/d^2$, is singular with probability at most d^{-2c} for a universal constant c . However, we need the probability of being singular to be of the order $d^{-\bar{C}}$ for some $\bar{C}$ which we will determine later in the proof. To achieve this smaller probability, consider drawing independently $\mathbf{E}_i, i \in [\frac{\bar{C}}{c}]$ $\text{Bern}(\rho)$ matrices. Then the probability of all of them being singular is at most $d^{-2\bar{C}}$. It is straightforward to see that whenever the determinant of a Boolean matrix is non-zero then the Boolean Determinant of that matrix is also non-zero. Hence, with probability at least $1 - d^{-2\bar{C}}$, at least one of these matrices has non-zero Boolean determinant. We create our matrix $\mathbf{M}' = \mathbf{E}_1 \vee \dots \vee \mathbf{E}_{\frac{\bar{C}}{c}}$, where the 'or' operation is done element-wise. Note that $\mathbf{M}'$ has a non-zero Boolean determinant if any of the matrices $\mathbf{E}_1 \dots \mathbf{E}_{\frac{\bar{C}}{c}}$ has a non-zero one. Thus $\mathbf{M}'$ has a non-zero Boolean determinant with probability at least $1 - d^{-2\bar{C}}$. Note that $\mathbf{M}'$ is itself an i.i.d.

Bernoulli matrix, with each element being $Bern(q')$ with $q' \leq \rho\bar{C} = \bar{C} \log d^2/d^2$. Hence we can set $q = \bar{C} \log d^2/d^2$.

Since M' has a zero Boolean determinant with probability at most $d^{-2\bar{C}}$, the probability that $\text{Det}_{\text{Bool}}(M_1 \odot M_2) = 0$ is also smaller than $d^{-2\bar{C}}$, and hence $W_1 \odot W_2$ are invertible with probability at least $1 - d^{-2\bar{C}}$. Then to ensure that the Khatri-Rao product of the sparse matrices is invertible for all the $2l$ layers of f , we need to do a union bound over all the layers. Thus, the probability that any of the Khatri-Rao products are not invertible is at most $d^{-2\bar{C}}2l$. Since we have assumed that $l = \mathcal{O}(\text{poly}(d))$, then for a large enough constant $\bar{C}$, the overall probability of the union bound can be driven down to be at most $1/d$. Recalling that $p = \sqrt{2dq}$, we get that $p = \Theta(\sqrt{\log d/d})$ suffices for the theorem to hold. $\square$

E.5 Experiments

Figure E.2: Verifying our constructions for BatchNorm parameters on data created by a random neural network g . The baseline algorithm (Blue curve) corresponds training the BatchNorm parameters of the network f_1 on this data using SGD, while the other parameters (weight matrices) are kept frozen at initialization. The green curve represents the network f_2 created by our construction by directly using the teacher network g . We also train a dense neural network g' with same architecture as g using SGD (dashed black curve). We then use our construction on g' to create f_3 , which is represented by the orange curve. In the figure on the left, we vary the width of the hidden layer of g , and set the width of f_1 , f_2 , and f_3 to be the one proposed in Theorem 9.5. In the figure on the right, we fix the width of g to be 25, and vary the widths of f_1 , f_2 , and f_3 .

To verify our construction numerically, we designed the following experiment: We create a random target network $g(x)$ with one hidden layer of width d , with ReLU activations. The output of the target network is simply the sum of outputs of the hidden layer neurons. We try to learn this function using the network $f(x)$ from Theorem 9.5 employing three algorithms:

- **Baseline:** We freeze the weight parameters of f and train only the BatchNorm parameters using SGD.
- **Constructive algorithm, directly from target network:** We freeze the weight parameters of f and set the BatchNorm parameters according to the construction prescribed in the proof of Theorem 9.5.
- **Constructive algorithm, from a dense learnt network:** For this approach, we assume that we do not have access to the network g but only to the input/output pairs

produced by it. Thus, we first take another randomly initialized network g' with the same architecture as g , and train it using SGD on the data generated by g . Then, we create f by freezing its weight parameters and setting the BatchNorm parameters according to the construction prescribed in the proof of Theorem 9.5, but using g' in the construction instead of g .

The results are shown in Figure E.2 (left). We see that both the constructive algorithms beat the baseline. These show that training BatchNorm parameters using SGD might not be the best algorithm. While the constructive algorithm which uses the target network directly might not be viable to use in practice (since the parameters of g are not available), the constructive algorithm that first trains a dense network and uses that to construct the BatchNorm parameters shows that there can be other, indirect ways of training BatchNorm parameters, which are better than simply using SGD.

To see the dependence of error on the width of f for the three algorithms, we fixed the width of g to be $d = 25$, and repeated the first experiment with varying widths of f . The results for this experiment are shown in Figure E.2 (right). Here we see that while at smaller widths the baseline (SGD) performs better (still only achieves a constant loss), as the width increases, our constructive algorithms beat SGD. Note that here we used pseudo-inverse for setting the BatchNorm parameters of f using our construction, since at lower widths, the matrices that need to be inverted become rank-deficient.

- (a) Sparsity = 0.2 (Weight matrices have 20% non-zero weights)
- (b) Sparsity = 0.05 (Weight matrices have 5% non-zero weights)

Figure E.3: For the same setting as in Figure E.2, we set the sparsity of the frozen weights of f_1 , f_2 , and f_3 to be 20% and 5%. The figures above show the results for these experiments. (Bottom left) We see that if the network is very sparse, and the width is also small, then SGD beats our construction.

We tested the three algorithms when the networks f_1 , f_2 , and f_3 were constrained to have sparse weights. The results are shown in Figure E.3. In the plot on the left in Subfigure E.3a, we see that when the dimensions are low as compared to the sparsity, the orange and the black curve are far; as well as the green curve has high error. However, as we increase the dimensions keeping the sparsity fixed, the orange and black curves come closer and the green curve also achieves almost 0 error. This agrees with the results of Theorem 9.9. However, we also see that when the network is extremely sparse (sparsity=0.05), none of the algorithms achieve small loss. However, SGD (blue curve) still achieves much better loss than the constructive algorithms.

Experiment Details. The training set and the test set each consisted of 1 million samples with random Gaussian inputs and the labels were set according to the teacher network. The experiments were repeated 5 times for each method, and the smallest and the largest losses were discarded while computing the error bars for the figures. For learning BatchNorm params using SGD (blue curve), three learning rate schedulers were tried: Cosine annealing scheduler, exponential decay scheduler, and constant learning rate, out of which constant learning rate performed the best. The experiments were run on a machine with Intel i9-9820X CPU with 131 GB RAM and GeForce RTX 2080 Ti GPU with 11GB RAM. The code can be found here - https://anonymous.4open.science/r/batch-norm_git-8F5B/.

F APP: LOGARITHMIC BOUNDS FOR STRONG LTH VIA SUBSET SUM PROBLEM

F.1 Proof of the upper bound

Complete proof of the Theorem 10.3 In the following subsections, we hierarchically build the construction for our proof of Theorem 10.3. We have shown how we approximate a single weight in Subsection 10.3.2. This first step is slightly different than the sketch above, in the sense that we approximate a single weight with a ReLU random network, rather than a linear one. We then approximate a single ReLU neuron in Subsection F.1.1, and a single layer in Subsection F.1.2. Finally, we approximate the whole network in Subsection F.1.3, which completes the proof of Theorem 10.3.

F.1.1 Approximating a single neuron

In this subsection we prove the following lemma on approximating a (univariate) linear function $\mathbf{w}^T \mathbf{x}$, which highlights the main idea in approximating a (multivariate) linear function $\mathbf{W}\mathbf{x}$ (see Lemma F.2 in Subsection F.1.2).

Lemma F.1. (*Approximating a univariate linear function*) Consider a randomly initialized neural network $g(\mathbf{x}) = \mathbf{v}^T \sigma(\mathbf{M}\mathbf{x})$ with $\mathbf{x} \in \mathbb{R}^d$ such that $\mathbf{M} \in \mathbb{R}^{Cd \log \frac{d}{\epsilon} \times d}$ and $\mathbf{v} \in \mathbb{R}^{Cd \log \frac{d}{\epsilon}}$, where each weight is initialized independently from the distribution $U[-1, 1]$.

Let $\hat{g}(\mathbf{x}) = (\mathbf{s} \odot \mathbf{v})^T \sigma((\mathbf{T} \odot \mathbf{M})\mathbf{x})$ be the pruned network for a choice of binary vector $\mathbf{s}$ and matrix $\mathbf{T}$. If $f_{\mathbf{w}}(\mathbf{x}) = \mathbf{w}^T \mathbf{x}$ be the linear function, then with probability at least $1 - \epsilon$,

$$\forall \mathbf{w} : \|\mathbf{w}\|_{\infty} \leq 1, \exists \mathbf{s}, \mathbf{T} : \sup_{\mathbf{x} : \|\mathbf{x}\|_{\infty} \leq 1} \|f_{\mathbf{w}}(\mathbf{x}) - \hat{g}(\mathbf{x})\| < \epsilon.$$

Proof. We will approximate $\mathbf{w}^T \mathbf{x}$ coordinate-wise. See Figure F.1 for illustration.

Step 1: Pre-processing M We first begin by pruning $\mathbf{M}$ to create a block-diagonal matrix $\mathbf{M}'$. Specifically, we create $\mathbf{M}'$ by only keep the following non-zero entries:

$$\mathbf{M}' = \begin{bmatrix} \mathbf{u}_1 & 0 & \dots & 0 \\ 0 & \mathbf{u}_2 & \dots & 0 \\ \vdots & \vdots & \dots & 0 \\ 0 & 0 & \dots & \mathbf{u}_d \end{bmatrix}, \quad \text{where } \mathbf{u}_i \in \mathbb{R}^{C \log(\frac{d}{\epsilon})}$$

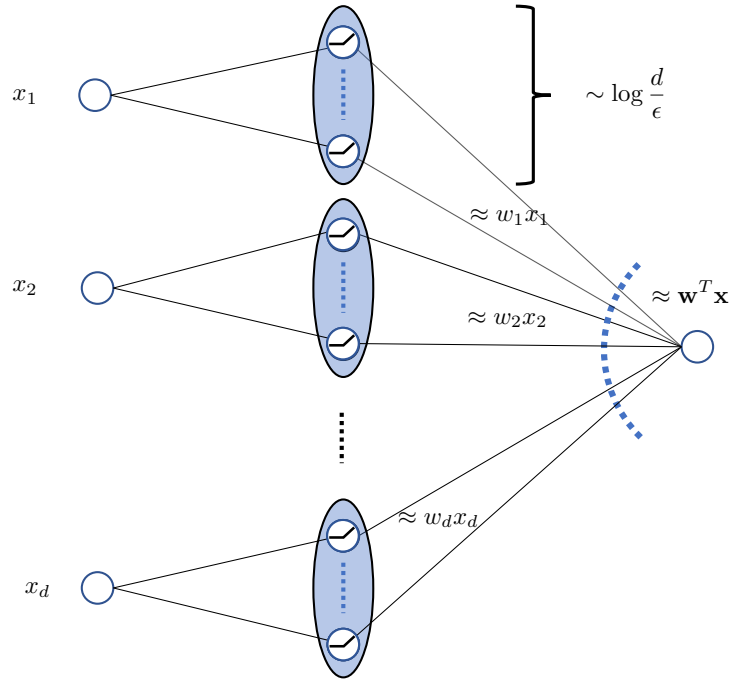


Figure F.1: Approximating a single neuron $\sigma(\mathbf{w}^T \mathbf{x})$: A diagram showing our construction to approximate a single neuron $\sigma(\mathbf{w}^T \mathbf{x})$. We construct the first hidden layer with d blocks (shown in blue), where each block contains $k = O\left(\log \frac{d}{\epsilon}\right)$ neurons. We first pre-process the weights by pruning the first layer so that it has a block structure as shown. For ease of visualization, we only show two connections per block, i.e., each neuron in the i^{th} block is connected to x_i and (before pruning) the output neuron. We then use Lemma 10.7 to show that second layer can be pruned so that i^{th} block approximates $w_i x_i$. Overall, the construction approximates $\mathbf{w}^T \mathbf{x}$. Note that, after an initial pre-processing of the first layer, we only prune the second layer so that we can re-use the weights to approximate other neurons in a layer.

We choose the binary matrix $\mathbf{T}$ to be such that $\mathbf{M}' = \mathbf{T} \odot \mathbf{M}$. We also decompose $\mathbf{v}$ and $\mathbf{s}$ as

$$\mathbf{s} = \begin{bmatrix} \mathbf{s}_1 \\ \mathbf{s}_2 \\ \vdots \\ \mathbf{s}_d \end{bmatrix}, \quad \mathbf{v} = \begin{bmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \\ \vdots \\ \mathbf{v}_d \end{bmatrix}, \quad \text{where } \mathbf{s}_i, \mathbf{v}_i \in \mathbb{R}^{C \log(\frac{d}{\epsilon})}.$$

Using this notation, we can express our network as the following:

$$(\mathbf{s} \odot \mathbf{v})^T \sigma(\mathbf{M}' \mathbf{x}) = \sum_{i=1}^d (\mathbf{s}_i \odot \mathbf{v}_i)^T \sigma(\mathbf{u}_i x_i). \quad (\text{F.1})$$

Step 2: Pruning u Let $n = C \log(d/\epsilon)$ and define the event $E_{i,\epsilon}$ be the following event from the Lemma 10.7:

$$E_{i,\epsilon} := \left\{ \sup_{w \in [-1,1]} \inf_{\mathbf{s}_i \in \{0,1\}^n} \sup_{x: |x| \leq 1} |wx - (\mathbf{v}_i \odot \mathbf{s}_i)^T \sigma(\mathbf{u}_i x)| \leq \epsilon \right\}$$

Define the event $E_\epsilon := \bigcap_i E_{i,\epsilon}$, the intersection of all the events. We consider the event $E_{\frac{\epsilon}{d}}$, where the approximation parameter is $\frac{\epsilon}{d}$. For each i , Lemma 10.7 shows that event $E_{i,\frac{\epsilon}{d}}$ holds with probability at least $1 - \frac{\epsilon}{d}$ because the dimension of $\mathbf{v}_i$ and $\mathbf{u}_i$ is at least $C \log(d/\epsilon)$. Taking a union bound we get that the event $E_{\frac{\epsilon}{d}}$ holds with probability at least $1 - \epsilon$. On the event $E_{\frac{\epsilon}{d}}$, we obtain the following series of inequalities:

$$\begin{aligned} & \sup_{\|\mathbf{w}\|_\infty \leq 1} \inf_{\mathbf{s}, \mathbf{T}} \sup_{\|\mathbf{x}\|_\infty \leq 1} |\mathbf{w}^T \mathbf{x} - (\mathbf{s}_2 \odot \mathbf{v})^T \sigma((\mathbf{S}_1 \odot \mathbf{M})\mathbf{x})| \\ & \leq \sup_{\|\mathbf{w}\|_\infty \leq 1} \inf_{\mathbf{s} \in \{0,1\}^{dn}} \sup_{\|\mathbf{x}\|_\infty \leq 1} |\mathbf{w}^T \mathbf{x} - (\mathbf{s}_2 \odot \mathbf{v})^T \sigma(\mathbf{M}'\mathbf{x})| \\ & \quad \text{(Pruning M according to Step 1 (Pre-processing M).)} \\ & = \sup_{\|\mathbf{w}\|_\infty \leq 1} \inf_{\mathbf{s}_1, \dots, \mathbf{s}_d \in \{0,1\}^n} \sup_{\|\mathbf{x}\|_\infty \leq 1} \left| \sum_{i=1}^d w_i x_i - \sum_{i=1}^d (\mathbf{s}_i \odot \mathbf{v}_i)^T \sigma(\mathbf{u}_i x_i) \right| \quad \text{(Using Eq. (F.1))} \\ & \leq \sup_{\|\mathbf{w}\|_\infty \leq 1} \inf_{\mathbf{s}_1, \dots, \mathbf{s}_d \in \{0,1\}^n} \sup_{\|\mathbf{x}\|_\infty \leq 1} \sum_{i=1}^d |w_i x_i - (\mathbf{s}_i \odot \mathbf{v}_i)^T \sigma(\mathbf{u}_i x_i)| \\ & = \sum_{i=1}^d \sup_{|w_i| \leq 1} \inf_{\mathbf{s}_i \in \{0,1\}^n} \sup_{|x_i| \leq 1} |w_i x_i - (\mathbf{s}_i \odot \mathbf{v}_i)^T \sigma(\mathbf{u}_i x_i)| \\ & \leq \sum_{i=1}^i d \frac{\epsilon}{d} \quad \text{(By definition of the event } E_{\frac{\epsilon}{d}}) \\ & \leq \epsilon. \end{aligned}$$

□

F.1.2 Approximating a single layer

In this subsection, we approximate a layer from the target network by pruning 2 layers of a randomly initialized network. The overview of the construction is given in Figure F.2.

Lemma F.2. (*Approximating a layer*) Consider a randomly initialized two layer neural network $g(\mathbf{x}) = \mathbf{N}\sigma(\mathbf{M}\mathbf{x})$ with $\mathbf{x} \in \mathbb{R}^{d_1}$ such that $\mathbf{N}$ has dimension $(d_2 \times C d_1 \log \frac{d_1 d_2}{\epsilon})$ and $\mathbf{M}$ has dimension $(C d_1 \log \frac{d_1 d_2}{\epsilon} \times d_1)$, where each weight is initialized independently from the distribution $U[-1, 1]$.

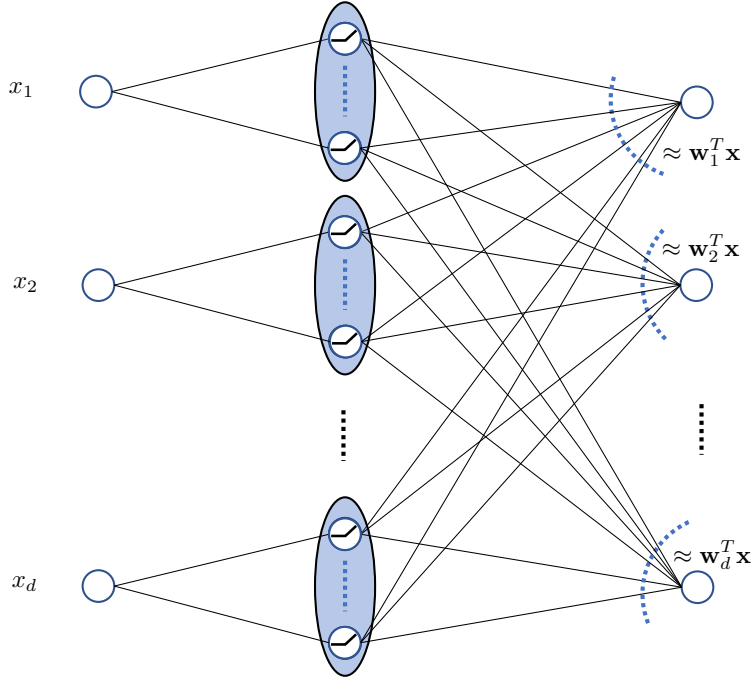


Figure F.2: Approximating a layer $\sigma(\mathbf{W}\mathbf{x})$: A diagram showing our construction to approximate a layer. Let $\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_d$ be the d rows of $\mathbf{W}$, i.e., the weights of d neurons. Our construction has an additional hidden layer, which contains d blocks (highlighted in blue), where each unit contains $k = O(\log(\frac{d}{\epsilon}))$ neurons. We first pre-process the weights by pruning the first layer so that it has a block structure as shown. For ease of visualization, we only show two connections per block, i.e., each neuron in the i^{th} block is connected to x_i and (before pruning) all the output neurons.

Let $\hat{g}(x) = (\mathbf{S} \odot \mathbf{N})^T \sigma((\mathbf{T} \odot \mathbf{M})\mathbf{x})$ be the pruned network for a choice of pruning matrices $\mathbf{S}$ and $\mathbf{T}$. If $f_{\mathbf{W}}(\mathbf{x}) = \mathbf{W}\mathbf{x}$ is the linear (single layered) network, where $\mathbf{W}$ has dimensions $d_2 \times d_1$, then with probability at least $1 - \epsilon$,

$$\sup_{\mathbf{W}: \|\mathbf{W}\| \leq 1, \mathbf{W} \in \mathbb{R}^{d_2 \times d_1}} \exists \mathbf{S}, \mathbf{T} : \sup_{\mathbf{x}: \|\mathbf{x}\|_{\infty} \leq 1} \|f_{\mathbf{W}}(\mathbf{x}) - \hat{g}(\mathbf{x})\| < \epsilon.$$

Proof. Our proof strategy is similar to the proof in Lemma F.1.

Step 1: Pre-processing M Similar to Lemma F.1, we begin by pruning M to get a block diagonal matrix $\mathbf{M}'$.

$$\mathbf{M}' = \begin{bmatrix} \mathbf{u}_1 & 0 & \dots & 0 \\ 0 & \mathbf{u}_2 & \dots & 0 \\ \vdots & \vdots & \dots & 0 \\ 0 & 0 & \dots & \mathbf{u}_{d_1} \end{bmatrix}, \quad \text{where } \mathbf{u}_i \in \mathbb{R}^{C \log(\frac{d_1 d_2}{\epsilon})}$$

Thus, $\mathbf{T}$ is such that $\mathbf{M}' = \mathbf{T} \odot \mathbf{M}$. We also decompose $\mathbf{N}$ and $\mathbf{S}$ as following

$$\mathbf{S} = \begin{bmatrix} \mathbf{s}_{1,1}^T & \dots & \mathbf{s}_{1,d_1}^T \\ \mathbf{s}_{2,1}^T & \dots & \mathbf{s}_{2,d_1}^T \\ \vdots & \dots & \vdots \\ \mathbf{s}_{d_2,1}^T & \dots & \mathbf{s}_{d_2,d_1}^T \end{bmatrix}, \quad \mathbf{N} = \begin{bmatrix} \mathbf{v}_{1,1}^T & \dots & \mathbf{v}_{1,d_1}^T \\ \mathbf{v}_{2,1}^T & \dots & \mathbf{v}_{2,d_1}^T \\ \vdots & \dots & \vdots \\ \mathbf{v}_{d_2,1}^T & \dots & \mathbf{v}_{d_2,d_1}^T \end{bmatrix}, \quad \text{where } \mathbf{v}_{i,j}, \mathbf{u}_i \in \mathbb{R}^{C \log(\frac{d_1 d_2}{\epsilon})}$$

Using this notation, we get the following relation:

$$(\mathbf{S} \odot \mathbf{N})\sigma(\mathbf{M}'\mathbf{x}) = \begin{bmatrix} \sum_{j=1}^{d_1} (\mathbf{s}_{1,j} \odot \mathbf{v}_{1,j})^T \sigma(\mathbf{u}_j x_j) \\ \vdots \\ \sum_{j=1}^{d_1} (\mathbf{s}_{d_2,j} \odot \mathbf{v}_{d_2,j})^T \sigma(\mathbf{u}_j x_j) \end{bmatrix} \quad (\text{F.2})$$

Step 2: Pruning N Note that $\mathbf{v}_{i,j}$ and $\mathbf{u}_i$ contain i.i.d. random variables from Uniform distribution. Let $n = C \log(d_1 d_2 / \epsilon)$ and define $E_{i,j,\epsilon}$ be the following event from the Lemma 10.7:

$$E_{i,j,\epsilon} := \left\{ \sup_{w \in [-1,1]} \inf_{\mathbf{s}_{i,j} \in \{0,1\}^n} \sup_{x: |x| \leq 1} |wx - (\mathbf{v}_{i,j} \odot \mathbf{s}_{i,j})^T \sigma(\mathbf{u}_i x)| \leq \epsilon \right\}$$

Define $E_\epsilon := \bigcap_{1 \leq i \leq d_2} \bigcap_{1 \leq j \leq d_1} E_{i,j,\epsilon}$ to be the intersection of all individual events. Lemma 10.7 states that each event $E_{i,j,\frac{\epsilon}{d_1 d_2}}$ holds with probability $1 - \frac{\epsilon}{d_1 d_2}$ because $\mathbf{u}_i$ and $\mathbf{v}_{i,j}$ have dimensions at least $C \log(\frac{d_1 d_2}{\epsilon})$. By a union bound, the event $E_{\frac{\epsilon}{d_1 d_2}}$ holds with probability $1 - \epsilon$. On the event $E_{\frac{\epsilon}{d_1 d_2}}$, we get the following inequalities:

$$\begin{aligned} & \sup_{\mathbf{w}: \|\mathbf{w}\| \leq 1} \inf_{\mathbf{S}, \mathbf{T}} \sup_{\|\mathbf{x}\|_\infty \leq 1} \|\mathbf{W}\mathbf{x} - (\mathbf{S} \odot \mathbf{N})^T \sigma((\mathbf{T} \odot \mathbf{M})\mathbf{x})\| \\ & \leq \sup_{\mathbf{w}: \|\mathbf{w}\| \leq 1} \inf_{\mathbf{S}} \sup_{\|\mathbf{x}\|_\infty \leq 1} \|\mathbf{W}\mathbf{x} - (\mathbf{S}_2 \odot \mathbf{N})^T \sigma(\mathbf{M}'\mathbf{x})\| \\ & \quad (\text{Pruning M according to Step 1 (Pre-processing M)}) \end{aligned}$$

$$\begin{aligned}
&\leq \sup_{\mathbf{w}: \|\mathbf{w}\| \leq 1} \inf_{\mathbf{s}_{i,j} \in \{0,1\}^n} \sup_{\|\mathbf{x}\|_\infty \leq 1} \sum_{i=1}^{d_2} \left| \sum_{j=1}^{d_1} w_{i,j} x_j - \sum_{j=1}^{d_1} (\mathbf{s}_{i,j} \odot \mathbf{v}_{i,j})^T \sigma(\mathbf{u}_j x_j) \right| \\
&\quad \text{(Using Eq. (F.2))} \\
&\leq \sup_{w_{i,j}: |w_{i,j}| \leq 1} \inf_{\mathbf{s}_{i,j} \in \{0,1\}^n} \sup_{x_j: |x_j| \leq 1} \sum_{i=1}^{d_2} \sum_{j=1}^{d_1} |w_{i,j} x_j - (\mathbf{s}_{i,j} \odot \mathbf{v}_{i,j})^T \sigma(\mathbf{u}_j x_j)| \\
&\leq \sup_{w_{i,j}: |w_{i,j}| \leq 1} \inf_{\mathbf{s}_{i,j} \in \{0,1\}^n} \sum_{i=1}^{d_2} \sum_{j=1}^{d_1} \sup_{x_j: |x_j| \leq 1} |w_{i,j} x_j - (\mathbf{s}_{i,j} \odot \mathbf{v}_{i,j})^T \sigma(\mathbf{u}_j x_j)| \\
&= \sum_{i=1}^{d_2} \sum_{j=1}^{d_1} \sup_{w_{i,j}: |w_{i,j}| \leq 1} \inf_{\mathbf{s}_{i,j} \in \{0,1\}^n} \sup_{x_j: |x_j| \leq 1} |w_{i,j} x_j - (\mathbf{s}_{i,j} \odot \mathbf{v}_{i,j})^T \sigma(\mathbf{u}_j x_j)| \\
&\leq d_1 d_2 \frac{\epsilon}{d_1 d_2} \leq \epsilon. \quad \text{(By definition of the event } E_{\frac{\epsilon}{d_1 d_2}})
\end{aligned}$$

□

F.1.3 Proof of Theorem 10.3

We now state the proof of Theorem 10.3 with the help of the lemmas in the previous subsection.

Proof. (Proof of Theorem 10.3) Let $\mathbf{x}_i$ be the input to the i -th layer of $f_{(\mathbf{w}_l, \dots, \mathbf{w}_1)}(\mathbf{x})$. Thus,

1. $\mathbf{x}_1 = \mathbf{x}$,
2. for $1 \leq i \leq l-1$, $\mathbf{x}_{i+1} = \sigma(\mathbf{W}_i \mathbf{x}_i)$.

Thus $f_{(\mathbf{w}_l, \dots, \mathbf{w}_1)}(\mathbf{x}) = \mathbf{W}_l \mathbf{x}_l$.

For i^{th} layer weights $\mathbf{W}_i$, let $\mathbf{S}_{2i}$ and $\mathbf{S}_{2i-1}$ be the binary matrices that achieve the guarantee in Lemma F.2. Lemma F.2 states that with probability $1 - \frac{\epsilon}{2l}$ the following event holds:

$$\begin{aligned}
&\sup_{\mathbf{w}_i \in \mathbb{R}^{d_{i+1} \times d_i}: \|\mathbf{w}_i\| \leq 1} \exists \mathbf{S}_{2i}, \mathbf{S}_{2i-1} : \sup_{\mathbf{x}: \|\mathbf{x}\| \leq 1} \|\mathbf{W}_i \mathbf{x} - (\mathbf{M}_{2i} \odot \mathbf{S}_{2i}) \sigma((\mathbf{S}_{2i} \odot \mathbf{M}_{2i-1}) \mathbf{x})\| \\
&\quad < \epsilon/2l.
\end{aligned} \tag{F.3}$$

As ReLU is 1-Lipschitz, the above event implies the following:

$$\begin{aligned}
&\sup_{\mathbf{w}_i \in \mathbb{R}^{d_{i+1} \times d_i}: \|\mathbf{w}_i\| \leq 1} \exists \mathbf{S}_{2i}, \mathbf{S}_{2i-1} : \sup_{\mathbf{x}: \|\mathbf{x}\| \leq 1} \|\sigma(\mathbf{W}_i \mathbf{x}) - \sigma((\mathbf{M}_{2i} \odot \mathbf{S}_{2i}) \sigma((\mathbf{S}_{2i} \odot \mathbf{M}_{2i-1}) \mathbf{x}))\| \\
&\quad < \epsilon/2l.
\end{aligned} \tag{F.4}$$

Taking a union bound, we get that with probability $1 - \epsilon$, the above inequalities (F.3) and (F.4) hold for every layer simultaneously. For the remainder of the proof, we will assume that this event holds. For the any fixed function f , let $g_f = g_{(\mathbf{w}_l, \dots, \mathbf{w}_1)}$ be the pruned network constructed layer-wise, by pruning with binary matrices satisfying Eq. (F.3) and Eq. (F.4), and let these pruned matrices be $\mathbf{M}'_i$. Let $\mathbf{x}'_i$ be the input to the $2i - 1$ -th layer of g_f . We note that $\mathbf{x}'_i$ satisfies the following recurrent relations:

1. $\mathbf{x}'_1 = \mathbf{x}$,
2. for $1 \leq i \leq l - 1$, $\mathbf{x}'_{i+1} = \sigma(\mathbf{M}'_{2i}\sigma(\mathbf{M}'_{2i-1}\mathbf{x}'_i))$.

Because the input $\mathbf{x}$ has $\|\mathbf{x}\| \leq 1$, Equation (F.4) also states that $\|\mathbf{x}'_i\| \leq \left(1 + \frac{\epsilon}{2l}\right)^{i-1}$. To see this, note that we use Equation (F.4) to get for $1 \leq i \leq l - 1$ as

$$\begin{aligned} \|\sigma(\mathbf{W}_i\mathbf{x}'_i) - \mathbf{x}'_{i+1}\| &\leq \|\mathbf{x}'_i\|(\epsilon/2l) \\ \implies \|\mathbf{x}'_{i+1}\| &\leq \|\mathbf{x}'_i\|(\epsilon/2l) + \|\sigma(\mathbf{W}_i\mathbf{x}'_i)\| \\ &\leq \|\mathbf{x}'_i\|(\epsilon/2l) + \|\mathbf{W}_i\mathbf{x}'_i\| \\ &\leq \|\mathbf{x}'_i\|(\epsilon/2l) + \|\mathbf{x}'_i\|. \end{aligned}$$

Applying this inequality recursively, we get the claim that for $1 \leq i \leq l - 1$, $\|\mathbf{x}'_i\| \leq \left(1 + \frac{\epsilon}{2l}\right)^{i-1}$. Using this, we can bound the error between $\mathbf{x}_i$ and $\mathbf{x}'_i$. For $1 \leq i \leq l - 1$,

$$\begin{aligned} \|\mathbf{x}_{i+1} - \mathbf{x}'_{i+1}\| &= \|\sigma(\mathbf{W}_i\mathbf{x}_i) - \sigma(\mathbf{M}'_{2i}\sigma(\mathbf{M}'_{2i-1}\mathbf{x}'_i))\| \\ &\leq \|\sigma(\mathbf{W}_i\mathbf{x}_i) - \sigma(\mathbf{W}_i\mathbf{x}'_i)\| + \|\sigma(\mathbf{W}_i\mathbf{x}'_i) - \sigma(\mathbf{M}'_{2i}\sigma(\mathbf{M}'_{2i-1}\mathbf{x}'_i))\| \\ &\leq \|\mathbf{x}_i - \mathbf{x}'_i\| + \|\mathbf{W}_i\mathbf{x}'_i - \mathbf{M}'_{2i}\sigma(\mathbf{M}'_{2i-1}\mathbf{x}'_i)\| \\ &< \|\mathbf{x}_i - \mathbf{x}'_i\| + \left(1 + \frac{\epsilon}{2l}\right)^{i-1} \frac{\epsilon}{2l}, \end{aligned}$$

where we use Equation (F.3). Unrolling this we get

$$\|\mathbf{x}_l - \mathbf{x}'_l\| \leq \sum_{i=1}^{l-1} \left(1 + \frac{\epsilon}{2l}\right)^{i-1} \frac{\epsilon}{2l}.$$

Finally using the inequality above, we get that with probability at least $1 - \epsilon$,

$$\begin{aligned} \|f_{(\mathbf{w}_l, \dots, \mathbf{w}_1)}(\mathbf{x}) - g_{(\mathbf{w}_l, \dots, \mathbf{w}_1)}(\mathbf{x})\| &= \|\mathbf{W}_l\mathbf{x}_l - \mathbf{M}'_{2l}\sigma(\mathbf{M}'_{2l-1}\mathbf{x}'_l)\| \\ &\leq \|\mathbf{W}_l\mathbf{x}_l - \mathbf{W}_l\mathbf{x}'_l\| + \|\mathbf{W}_l\mathbf{x}'_l - \mathbf{M}'_{2l}\sigma(\mathbf{M}'_{2l-1}\mathbf{x}'_l)\| \\ &\leq \|\mathbf{x}_l - \mathbf{x}'_l\| + \|\mathbf{W}_l\mathbf{x}'_l - \mathbf{M}'_{2l}\sigma(\mathbf{M}'_{2l-1}\mathbf{x}'_l)\| \end{aligned}$$

$$\begin{aligned}
&< \|\mathbf{x}_l - \mathbf{x}'_l\| + \left(1 + \frac{\epsilon}{2l}\right)^{l-1} \frac{\epsilon}{2l} \\
&\leq \left(\sum_{i=1}^{l-1} \left(1 + \frac{\epsilon}{2l}\right)^{i-1} \frac{\epsilon}{2l}\right) + \left(1 + \frac{\epsilon}{2l}\right)^{l-1} \frac{\epsilon}{2l} \\
&\leq \sum_{i=1}^l \left(1 + \frac{\epsilon}{2l}\right)^{i-1} \frac{\epsilon}{2l} \\
&= \left(1 + \frac{\epsilon}{2l}\right)^l - 1 \\
&< e^{\epsilon/2} - 1 \\
&< \epsilon. \tag{Since $\epsilon < 1$.}
\end{aligned}$$

Replacing ϵ in this proof with $\min\{\epsilon, \delta\}$ gives us the statement of the theorem. $\square$

F.2 Proof of Lower Bound

Proof. (Proof of Theorem 10.8) Firstly, note that $h_{\mathbf{W}}(\mathbf{x}) = \mathbf{W}\mathbf{x}$. Another fact we use in this proof is that matrices $\mathbf{W}$ of dimension $d \times d$ can be considered as points in the space $\mathbb{R}^{d \times d} \equiv \mathbb{R}^{d^2}$. The metric that we would be using on this space would be the operator norm of matrices $\|\cdot\|$. Note that $\mathcal{G}$ is a random set of functions, but we abuse the notation by using $|\mathcal{G}|$ denote the *maximum* number of sub-networks that can be formed, starting from any initialization with the given architecture.

Step 1: Packing argument. Consider the normed space of $d \times d$ matrices, $\mathcal{W} = \{\mathbf{W} \in \mathbb{R}^{d \times d} : \|\mathbf{W}\| \leq 1\}$, with the operator norm $\|\cdot\|$. Let $\mathcal{P}$ be a 2ϵ -separated set of $(\mathcal{W}, \|\cdot\|)$, i.e. $\mathcal{P} \subset \mathcal{W}$ and $\|\mathbf{M} - \mathbf{M}'\| > 2\epsilon$ for all distinct $\mathbf{M}, \mathbf{M}' \in \mathcal{P}$.

Note that any function g' can only approximate at most one member of $\mathcal{P}$. To see this, let us assume on the contrary that a g' can approximate two distinct members $\mathbf{W}_1$ and $\mathbf{W}_2$ of $\mathcal{P}$. Then a triangle inequality states that

$$\begin{aligned}
\|\mathbf{W}_1 - \mathbf{W}_2\| &= \sup_{\mathbf{x}: \|\mathbf{x}\| \leq 1} \|\mathbf{W}_1\mathbf{x} - \mathbf{W}_2\mathbf{x}\| \\
&\leq \sup_{\mathbf{x}: \|\mathbf{x}\| \leq 1} \|g'(\mathbf{x}) - \mathbf{W}_1\mathbf{x}\| + \sup_{\mathbf{x}: \|\mathbf{x}\| \leq 1} \|g'(\mathbf{x}) - \mathbf{W}_2\mathbf{x}\| \leq 2\epsilon,
\end{aligned}$$

which is a contradiction to the definition of a 2ϵ -separated set. Hence, g' can approximate at most only one member of $\mathcal{P}$.

Step 2: Relation between $|\mathcal{G}|$ and $|\mathcal{P}|$. The goal of this step is to show that, under the theorem assumptions, $|\mathcal{P}| < 2|\mathcal{G}|$. If $|\mathcal{P}| > 2|\mathcal{G}|$, then we show that one of the matrices in $\mathcal{P}$ is the difficult matrix W that we're looking for.

Let us assume that $|\mathcal{P}| > 2|\mathcal{G}|$. Recall that the previous step states that, for any realization of g , the corresponding $\mathcal{G}$ can only approximate at most $|\mathcal{G}|$ matrices in $\mathcal{P}$. Therefore, for a fixed realization of $\mathcal{G}$, we get that

$$\frac{\sum_{\mathbf{W} \in \mathcal{P}} \mathbb{I}(\exists g' \in \mathcal{G} : \sup_{\mathbf{x}: \|\mathbf{x}\| \leq 1} \|g'(\mathbf{x}) - \mathbf{W}\mathbf{x}\| \leq \epsilon)}{|\mathcal{P}|} \leq \frac{|\mathcal{G}|}{|\mathcal{P}|} < \frac{1}{2}.$$

Taking the expectation over the distribution of g , we get that

$$\frac{\sum_{\mathbf{W} \in \mathcal{P}} \mathbb{P}(\exists g' \in \mathcal{G} : \sup_{\mathbf{x}: \|\mathbf{x}\| \leq 1} \|g'(\mathbf{x}) - W\mathbf{x}\| \leq \epsilon)}{|\mathcal{P}|} < \frac{1}{2}.$$

As the minimum is less than the average, there exists a $\mathbf{W} \in \mathcal{P}$ such that

$\mathbb{P}(\exists g' \in \mathcal{G} : \sup_{\mathbf{x}: \|\mathbf{x}\| \leq 1} \|g'(\mathbf{x}) - W\mathbf{x}\| \leq \epsilon) < \frac{1}{2}$, which is a contradiction to Eq. (10.9). Therefore, $2|\mathcal{G}| > |\mathcal{P}|$.

Step 3: Lower bound on $|\mathcal{P}|$. We will now choose $\mathcal{P}$ with the maximum cardinality of all 2ϵ -separated sets, i.e., that achieves the packing number. As packing number is lower bounded by the covering number, we will try to find a lower bound on the size of an 2ϵ -net of $\mathcal{W}$ (Vershynin, 2018b, Lemma 4.2.8). Now, any 2ϵ -cover has to have at least $\frac{\text{Vol}(\{\mathbf{W}: \|\mathbf{W}\| \leq 1\})}{\text{Vol}(\{\mathbf{W}: \|\mathbf{W}\| \leq 2\epsilon\})}$ elements, where the volume is the Lebesgue measure in $\mathbb{R}^{d \times d} = \mathbb{R}^{d^2}$. We also have that $\text{Vol}(\{\mathbf{W} : \|\mathbf{W}\| \leq c\}) > 0$ because $\{\mathbf{W} : \|\mathbf{W}\| \leq c\}$ contains $\{\mathbf{W} : \|\mathbf{W}\|_{\text{Frobenius}} \leq c\}$. Thus, we get that $\frac{\text{Vol}(\{\mathbf{W}: \|\mathbf{W}\| \leq 1\})}{\text{Vol}(\{\mathbf{W}: \|\mathbf{W}\| \leq 2\epsilon\})} = (2\epsilon)^{-d^2}$. Putting everything together, we get that

$$2|\mathcal{G}| > |\mathcal{P}| > |\mathcal{N}(\mathcal{W}, \|\cdot\|, 2\epsilon)| \geq \left(\frac{1}{2\epsilon}\right)^{-d^2}.$$

Case $l = 2$ Let the dimension of $\mathbf{M}_2$ be $d \times s$ and the dimension of $\mathbf{M}_1$ be $s \times d$. We need a lower bound on s . Now, the number of matrices that can be created by pruning $\mathbf{M}_2$ are 2^{sd} and similarly the number of matrices that can be created by pruning $\mathbf{M}_1$ are 2^{sd} . Thus, the total number of ReLUs that can be formed by pruning $\mathbf{M}_2$ and $\mathbf{M}_1$ is at most 2^{2sd} . Thus, $|\mathcal{G}| \leq 2^{2sd}$. Therefore, we get that

$$2^{2sd+1} > \left(\frac{1}{2\epsilon}\right)^{-d^2}.$$

This shows that $s = \Omega\left(d \log\left(\frac{1}{2\epsilon}\right)\right)$ is needed to approximate every function in $\mathcal{F}$ by pruning g with probability $1/2$.

Case $l > 2$ Let the total number of parameters be m . Therefore, we get that $|\mathcal{G}| \leq 2^m$. Following the same arguments as before, we get that $m = \Omega\left(d^2 \log\left(\frac{1}{2\epsilon}\right)\right)$. $\square$

F.3 Subset sum results

F.3.1 Product of uniform distributions contains a uniform distribution

Lemma F.3. *Let $X \sim U[0, 1]$ (or $X \sim U[-1, 0]$) and $Y \in U[-1, 1]$ be independent random variables. Then the PDF of the random variable XY is*

$$f_{XY}(z) = \begin{cases} \frac{1}{2} \log \frac{1}{|z|} & |z| \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

Proof. It is easy to see why $f_{XY}(z) = 0$ for $z > 1$. We prove for $X \sim U[0, 1]$. The proof for $X \sim U[-1, 0]$ is similar.

Let us first try to find the CDF of XY .

Let $0 \leq z \leq 1$ be a real number. Note that $XY \leq 1$. Now, if $XY \leq z$, and if $Y \geq z$, then $X \leq z/Y$. However, if $Y < z$, then X can be anything in its support $[0, 1]$. Thus,

$$\begin{aligned} F_{XY}(z) &= \mathbb{P}(XY \leq z) \\ &= \int_0^z \frac{1}{2} \int_0^1 1 dx dy + \int_z^1 \frac{1}{2} \int_0^{z/y} 1 dx dy \\ &= \frac{z}{2} + \frac{1}{2} \int_z^1 \frac{z}{y} dy \\ &= \frac{z}{2} - \frac{z \log z}{2}. \end{aligned}$$

Differentiating this, the pdf for $0 \leq z \leq 1$ is

$$f_{XY}(z) = \frac{1}{2} \log \frac{1}{z}.$$

Now, because XY is symmetric around 0, we get that for $|z| \leq 1$

$$f_{XY}(z) = \frac{1}{2} \log \frac{1}{|z|}.$$

□

Corollary F.4. *Let $X \sim U[0, 1]$ (or $X \sim U[-1, 0]$) and $Y \in U[-1, 1]$ be independent random variables. Let P be the distribution of XY . Let δ_0 be the Dirac-delta function. Define a distribution $D = \frac{1}{2}\delta_0 + \frac{1}{2}P$.*

Then, there exists a distribution Q such that

$$P = \left(\frac{1}{2} \log 2\right) U\left[-\frac{1}{2}, \frac{1}{2}\right] + \left(1 - \frac{1}{2} \log 2\right) Q$$

Proof. The corollary follows from the observation that Lemma F.3 shows that pdf of P is lower bounded by $(\log 2)U\left[-\frac{1}{2}, \frac{1}{2}\right]$ on $\left[-\frac{1}{2}, \frac{1}{2}\right]$. □

F.3.2 Subset sum problem with product of uniform distributions

Corollary F.5 (Lueker (1998b)). *Let $X_1, \dots, X_n$ be i.i.d. from the distribution in the hypothesis of Corollary F.4, where $n \geq C \log \frac{2}{\epsilon}$ (for some universal constant C). Then, with probability at least $1 - \epsilon$, we have*

$$\forall z \in [-1, 1], \quad \exists S \subset [n] \text{ such that } \left| z - \sum_{i \in S} X_i \right| \leq \epsilon.$$

Proof. This is a direct application of Markov's inequality on Corollary 3.3 from Lueker (1998b) applied to the distribution in the hypothesis of Corollary F.4. □

G APP: LOGARITHMIC BOUNDS FOR STRONG LTH FOR BINARY NETWORKS

G.1 Proof of Lemma 11.7

Before proving Lemma 11.7 in its entirety, we first extend Lemma 11.6 to approximate a neuron with $\log(d/\epsilon)$ -precision.

Lemma G.1. *Consider a network $h(\mathbf{x}) = \sigma(\mathbf{w}^T \mathbf{x})$ where $\mathbf{w} \in \mathbb{R}^d$, $\|\mathbf{w}\| \leq 1$ and $\epsilon > 0$. Let $\hat{\mathbf{w}}$ be a coordinate-wise finite-precision truncation of $\mathbf{w}$ up to $\log(d/\epsilon)$ digits and $\hat{g}(\mathbf{x}) = \sigma(\hat{\mathbf{w}}^T \mathbf{x})$. Then we have that*

$$\max_{\mathbf{x} \in \mathbb{R}^d: \|\mathbf{x}\| \leq 1} \|h(\mathbf{x}) - \hat{g}(\mathbf{x})\| \leq \epsilon.$$

Proof. Once again, by Lemma 11.5, we have that for each coordinate $|w_i - \hat{w}_i| \leq \epsilon/d$. Therefore, $\|\mathbf{w} - \hat{\mathbf{w}}\| \leq \sqrt{\sum_{i=1}^d \left(\frac{\epsilon}{d}\right)^2} \leq \epsilon$. Applying Cauchy-Schwarz with $\|\mathbf{x}\| \leq 1$ completes the proof. $\square$

We now extend the result to approximating a single layer with finite-precision.

Lemma G.2. *Consider a network $h(\mathbf{x}) = \sigma(\mathbf{1}^T \sigma(\mathbf{W}_1 \mathbf{x}))$ where $\mathbf{W}_1 \in \mathbb{R}^{d_1 \times d_0}$, $\|\mathbf{W}_1\| \leq 1$ and $\epsilon > 0$. Let $\widehat{\mathbf{W}}_1$ be a coordinate-wise finite-precision truncation of $\mathbf{W}_1$ up to $\log(d^2/\epsilon)$ digits and $\hat{g}(\mathbf{x}) = \sigma(\widehat{\mathbf{W}}_1^T \mathbf{x})$ where $d = \max\{d_0, d_1\}$. Then we have that*

$$\max_{\mathbf{x} \in \mathbb{R}^{d_0}: \|\mathbf{x}\| \leq 1} \|h(\mathbf{x}) - \hat{g}(\mathbf{x})\| \leq \epsilon.$$

Proof. Note that for any $\mathbf{x} : \|\mathbf{x}\| \leq 1$,

$$\begin{aligned} \|h(\mathbf{x}) - \tilde{g}(\mathbf{x})\| &= \left\| \sigma\left(\sum_{i=1}^{d_1} \sigma(W_1^{(i)} \mathbf{x})\right) - \sigma\left(\sum_{i=1}^{d_1} \sigma(\widehat{W}_1^{(i)} \mathbf{x})\right) \right\| \\ &\leq \left\| \sum_{i=1}^{d_1} \sigma(W_1^{(i)} \mathbf{x}) - \sum_{i=1}^{d_1} \sigma(\widehat{W}_1^{(i)} \mathbf{x}) \right\| && \text{(Since } \sigma \text{ is 1-Lipschitz)} \\ &\leq \sum_{i=1}^{d_1} \|\sigma(W_1^{(i)} \mathbf{x}) - \sigma(\widehat{W}_1^{(i)} \mathbf{x})\| \\ &\leq \sum_{i=1}^{d_1} \frac{\epsilon}{\max\{d_0, d_1\}} \\ &\leq \epsilon \end{aligned}$$

Since this holds for any $\mathbf{x}$, it also holds for the $\mathbf{x}$ that attains the maximum possible error which completes the proof. $\square$

We are now ready to prove Lemma 11.7 which states that to approximate any FC ReLU network within ϵ , it suffices to simply consider weights with logarithmic precision.

Lemma G.3. *Consider a network $h(\mathbf{x}) = \sigma(\mathbf{W}_l \sigma(\mathbf{W}_{l-1} \dots \sigma(\mathbf{W}_1 \mathbf{x})))$ where $\mathbf{W}_i \in \mathbb{R}^{d_i \times d_{i-1}}$, $\|\mathbf{W}_i\| \leq 1$, $d_i \leq d$ and $\epsilon > 0$. Let $\hat{g}(\mathbf{x}) = \sigma(\widehat{\mathbf{W}}_l \sigma(\widehat{\mathbf{W}}_{l-1} \dots \sigma(\widehat{\mathbf{W}}_1 \mathbf{x})))$ where $\widehat{\mathbf{W}}_i$ is a finite precision truncation of $\mathbf{W}_i$ up to $\log(d^2 l / \epsilon)$ digits. Then we have that*

$$\max_{\mathbf{x} \in \mathbb{R}^{d_0} : \|\mathbf{x}\| \leq 1} \|h(\mathbf{x}) - \hat{g}(\mathbf{x})\| \leq \epsilon.$$

Proof. The proof follows inductively. First note that since operator norm is bounded by the frobenius norm, we have for the output of the first layer that,

$$\begin{aligned} \|\sigma(\mathbf{W}_1 \mathbf{x}) - \sigma(\widehat{\mathbf{W}}_1 \mathbf{x})\| &\leq \|\mathbf{W}_1 \mathbf{x} - \widehat{\mathbf{W}}_1 \mathbf{x}\| \\ &\leq \|\mathbf{W}_1 - \widehat{\mathbf{W}}_1\| \\ &\leq \|\mathbf{W}_1 - \widehat{\mathbf{W}}_1\|_F \\ &\leq \epsilon/l \end{aligned}$$

Next, denote the output of the first layer by $\mathbf{x}_2 := \sigma(\mathbf{W}_1 \mathbf{x})$, $\hat{\mathbf{x}}_2 := \sigma(\widehat{\mathbf{W}}_1 \mathbf{x})$. Repeating the calculation above for the second layer gives us,

$$\begin{aligned} \|\sigma(\mathbf{W}_2 \sigma(\mathbf{W}_1 \mathbf{x})) - \sigma(\widehat{\mathbf{W}}_2 (\sigma(\widehat{\mathbf{W}}_1 \mathbf{x})))\| &= \|\sigma(\mathbf{W}_2 \mathbf{x}_2) - \sigma(\widehat{\mathbf{W}}_2 \hat{\mathbf{x}}_2)\| \\ &= \|\sigma(\mathbf{W}_2 \mathbf{x}_2) - \sigma(\widehat{\mathbf{W}}_2 \mathbf{x}_2) + \sigma(\widehat{\mathbf{W}}_2 \mathbf{x}_2) - \sigma(\widehat{\mathbf{W}}_2 \hat{\mathbf{x}}_2)\| \\ &\leq \|\sigma(\mathbf{W}_2 \mathbf{x}_2) - \sigma(\widehat{\mathbf{W}}_2 \mathbf{x}_2)\| + \|\sigma(\widehat{\mathbf{W}}_2 \mathbf{x}_2) - \sigma(\widehat{\mathbf{W}}_2 \hat{\mathbf{x}}_2)\| \end{aligned}$$

Note that the first term is bounded by ϵ/l since $\|\mathbf{x}_2\| \leq 1$. Further, using the fact that $\|\widehat{\mathbf{W}}_2\| \leq 1$ and $\|\mathbf{x}_2 - \hat{\mathbf{x}}_2\| \leq \epsilon/l$ as proved above, we get that

$$\|\sigma(\mathbf{W}_2 \sigma(\mathbf{W}_1 \mathbf{x})) - \sigma(\widehat{\mathbf{W}}_2 (\sigma(\widehat{\mathbf{W}}_1 \mathbf{x})))\| \leq 2\epsilon/l$$

By induction, for each layer $1 \leq i \leq l$ we have that for any $\mathbf{x} : \|\mathbf{x}\| \leq 1$,

$$\|(\sigma(\mathbf{W}_i \sigma(\mathbf{W}_{i-1} \dots \sigma(\mathbf{W}_1 \mathbf{x})))) - (\sigma(\widehat{\mathbf{W}}_i \sigma(\widehat{\mathbf{W}}_{i-1} \dots \sigma(\widehat{\mathbf{W}}_1 \mathbf{x}))))\| \leq \epsilon \cdot (i/l)$$

Setting $i = l$ completes the proof. $\square$

Lemma G.4. Consider a network $f(\mathbf{x}) = \sigma(\mathbf{W}_l \sigma(\mathbf{W}_{l-1} \dots \sigma(\mathbf{W}_1 \mathbf{x})))$ where $\mathbf{W}_i \in \mathbb{Z}^{d_i \times d_{i-1}}$, $\|\mathbf{W}_i\|_{\max} \leq W$ and $d_i \leq d$. Let $g(\mathbf{x})$ be a randomly initialized binary network of width $\Theta\left(d \log\left(\frac{dl \log^2 |W|}{\delta}\right)\right)$ and depth $\Theta(l \log |W|)$ such that every weight is drawn uniformly from $\{-1, +1\}$. Then $g(\mathbf{x})$ can be pruned to $\tilde{g}(\mathbf{x})$ so that $\tilde{g}(\mathbf{x}) = f(\mathbf{x})$ for all $\mathbf{x} \in \mathbb{R}^{d_0}$ with probability at least $1 - \delta$.

Proof. Consider any one particular diamond structure in Figure 11.2. If we choose the weights of this network uniformly at random from $\{-1, +1\}$, then the probability that they are all 1 is at most $(1/2)^4$. If we make the network k times wider, the probability that such a diamond-gadget does not exist in the network is given by $(1 - (1/2)^4)^k$. In other words, if we define the event A to be the failure event, then $P(A) = (1 - (1/2)^4)^k$. Note that there are $\Theta(dl \log^2 |W|)$ such diamond structures in our network. By symmetry, the failure probability of each of them is identical. To have the overall probability of failure to be at most δ , taking the union bound we have that

$$dl \log^2 |W| \cdot \left(1 - (1/2)^4\right)^k \leq \delta$$

Hence, it suffices to have $k \geq \Theta\left(\log\left(\frac{dl \log^2 |W|}{\delta}\right)\right)$. In other words, a randomly initialized binary network of width $\Theta\left(\log\left(\frac{dl \log^2 |W|}{\delta}\right)\right)$ and depth $\Theta(l \log |W|)$ contains our deterministic construction with probability at least $1 - \delta$. $\square$

G.2 Observation on the power of zero

We try to understand here if pruning is essential to the expressive power of binary networks. We note that pruning binary networks is strictly more powerful than merely sign-flipping their weights. To see this, consider the set of all networks that can be derived by pruning 1-hidden layer binary networks of width m and input dimension d :

$$\mathcal{F}_{pruned}^m = \left\{ f : f(\mathbf{x}) = \sum_{j=1}^m v_j \sigma\left(\sum_{i=1}^d w_j^i x_i\right), w_j^i, v_j \in \{+1, -1, 0\} \right\}.$$

Similarly, the set of all 1-hidden layer binary networks of the same width without pruning is given by

$$\mathcal{F}_{bin}^m = \left\{ f : f(\mathbf{x}) = \sum_{j=1}^m v_j \sigma\left(\sum_{i=1}^d w_j^i x_i\right), w_j^i, v_j \in \{+1, -1\} \right\}.$$

We prove the following proposition that shows that $\mathcal{F}_{pruned}$ is a strictly richer class of functions indicating that pruning is an essential part of approximating classifiers.

Proposition G.5. *The function $f(\mathbf{x}) = \sum_{j=1}^d \sigma\left(\sum_{i=1}^j x_i\right)$ satisfies $f(\mathbf{x}) \in \mathcal{F}_{pruned}^d$ and $f(\mathbf{x}) \notin \mathcal{F}_{bin}^d$, i.e., without pruning, $f(\mathbf{x})$ cannot be represented by a single layer binary network of width d .*

Proof. First, note that we recover $f(\mathbf{x})$ from $\mathcal{F}_{pruned}^d$ if we set $m = d$, $v_j = 1$ and $w_j^i = \mathbb{1}_{j \leq i}$ for all i, j . Therefore, $f(\mathbf{x}) \in \mathcal{F}_{pruned}^d$ holds.

To see that $f(\mathbf{x}) \notin \mathcal{F}_{bin}^d$, we evaluate $f(\cdot)$ at $\tilde{\mathbf{x}} = (x_1, 0, 0, \dots, 0, x_d)$ where $x_1 \geq x_d \geq 0$. Note that $f(\tilde{\mathbf{x}}) = \sigma(x_1) + d\sigma(x_d) = x_1 + d \cdot x_d$. Further, note that $\sigma(x_1 - x_d) = x_1 - x_d$, $\sigma(x_1 + x_d) = x_1 + x_d$ and $\sigma(-x_1 + x_d) = \sigma(-x_1 - x_d) = 0$. Therefore, for this specific $\tilde{\mathbf{x}}$, $\mathcal{F}_{bin}^d$ reduces to $\mathcal{F}_{bin}^d(\tilde{\mathbf{x}}) = \{(n_1 - n_2)(x_1 + x_d) + (n_3 - n_4)(x_1 - x_d)\} = \{(n_1 + n_2 - n_3 - n_4)x_1 + (n_1 - n_2 + n_3 - n_4)x_d\}$ where $n_1 + n_2 + n_3 + n_4 \leq d$, $n_1, n_2, n_3, n_4 \geq 0$. Clearly $f(\tilde{\mathbf{x}}) \notin \mathcal{F}_{bin}(\tilde{\mathbf{x}})$ for any valid n_1, n_2, n_3, n_4 . Therefore, it also holds that $f \notin \mathcal{F}_{bin}$. $\square$

Remark G.6. *The above proposition suggests that pruning is more powerful than merely flipping signs of a binary network. In fact, the same argument can be extended for binary networks of any fixed width d and depth l to show that pruned networks are more expressive. However, it does not quantify this difference in expressivity.*

H APP: EFFICIENT MEMORIZATION USING THRESHOLD NETWORKS

H.1 Proof of Theorem 12.8

Proof. As explained in Section 12.5, the network memorizes in four steps. In this proof, we will go into the details of each step.

Step 1: Generating unique binary representations. The first layer’s task is to generate unique binary representations for each sample in the dataset. What this means is that for every pair of samples $\mathbf{x}_i$ and $\mathbf{x}_j$, there should exist at least one neuron in the first layer, say $\sigma(\mathbf{w}^\top \mathbf{x} + b)$ such that $\sigma(\mathbf{w}^\top \mathbf{x}_i + b) \neq \sigma(\mathbf{w}^\top \mathbf{x}_j + b)$. Note that $\mathbf{w}^\top \mathbf{x} + b$ is just a hyperplane and $\sigma(\mathbf{w}^\top \mathbf{x}_i + b) \neq \sigma(\mathbf{w}^\top \mathbf{x}_j + b)$ is equivalent to saying that $\mathbf{x}_i$ and $\mathbf{x}_j$ lie on the opposite sides of the hyperplane. The next two lemmas says that we can easily find a set of $\mathcal{O}(\log(n)/\delta)$ hyperplanes such that for every pair of points $\mathbf{x}_i, \mathbf{x}_j$, at least one hyperplanes passes in between them.

Lemma H.1. Assume that our dataset satisfies Assumption 12.5. Define $m := \left\lceil \frac{4\pi}{\delta} \log \frac{n}{\epsilon} \right\rceil$ for any $\epsilon \in (0, 1)$. If we sample m hyperplanes of the form $\mathbf{w}_i^\top \mathbf{x} = 0$, where $\mathbf{w}_i \stackrel{i.i.d.}{\sim} \mathcal{N}(\mathbf{0}, \mathbf{I}_d)$, for $i = 1, \dots, m$, then with probability $1 - \epsilon$,

$$\forall i \neq j, \exists k : (\mathbf{w}_k^\top \mathbf{x}_i)(\mathbf{w}_k^\top \mathbf{x}_j) < 0.$$

Lemma H.2. Assume that our dataset satisfies Assumption 12.6. Define $m := \frac{C}{\delta} \log \left(\frac{n}{\epsilon} \right)$ for some universal constant C and any $\epsilon \in (0, 1)$. If we sample m hyperplanes of the form $\mathbf{w}_i^\top \mathbf{x} + \mathbf{b}_i = 0$, for $i = 1, \dots, m$, where $\mathbf{w}_i \stackrel{i.i.d.}{\sim} \mathcal{N}(\mathbf{0}, \mathbf{I}_d)$ and $\mathbf{b}_i \stackrel{i.i.d.}{\sim} \mathcal{N}(0, 1)$, then with probability $1 - \epsilon$,

$$\forall i \neq j, \exists k : (\mathbf{w}_k^\top \mathbf{x}_i + \mathbf{b}_k)(\mathbf{w}_k^\top \mathbf{x}_j + \mathbf{b}_k) < 0.$$

Note that each neuron in the first layer outputs either 0 or 1. If we concatenate all the outputs of these $\mathcal{O}(\log(n)/\delta)$ neurons, we will get a binary vector of length $\mathcal{O}(\log(n)/\delta)$. The lemmas above say that under Assumption 12.5 or Assumption 12.6, these binary vectors will be unique.

Let $\mathbf{z}_i^{(1)}$ be the binary vector created by the first layer when the input to the network is $\mathbf{x}_i$. Further, let $d^{(1)}$ denote the dimension (length) of this binary vector. Note that the first layer has $\mathcal{O}(\log(n)/\delta)$ neurons and $\mathcal{O}(d \log(n)/\delta)$ weights and biases.

Step 2: Compressing the binary representations. The next few layers compress the $\mathcal{O}(\log(n)/\delta)$ length binary vectors down to $\mathcal{O}(\log(n))$ length unique binary representations for each sample. We will use linear codes in $\mathbb{F}_2$ (the finite field of two elements: 0 and 1) for this.

For two binary vectors $\mathbf{b}_1, \mathbf{b}_2 \in \{0, 1\}^{d^{(1)}}$, define the inner product in the field as

$$\langle \mathbf{b}_1, \mathbf{b}_2 \rangle_{\oplus} := (b_{1,1} \cdot b_{2,1}) \oplus (b_{1,2} \cdot b_{2,2}) \oplus \cdots \oplus (b_{1,d^{(1)}} \cdot b_{2,d^{(1)}}),$$

where $\oplus$ is the XOR operator, and $b_{1,i}$ is the i -th element of $\mathbf{b}_1$, and similarly $b_{2,i}$ is the i -th element of $\mathbf{b}_2$.

Let $\mathbf{b}_1, \dots, \mathbf{b}_m$ be m i.i.d. Bernoulli(0.5) vectors of length $d^{(1)}$ each. Define vectors $\mathbf{z}_i^{(2)}$ as

$$\mathbf{z}_i^{(2)} = \begin{bmatrix} \langle \mathbf{b}_1, \mathbf{z}_i^{(1)} \rangle_{\oplus} \\ \vdots \\ \langle \mathbf{b}_m, \mathbf{z}_i^{(1)} \rangle_{\oplus} \end{bmatrix}, \forall i \in [n]. \quad (\text{H.1})$$

The next lemma says that we only need $m = \mathcal{O}(\log n)$ to ensure that $\forall i \neq j, \mathbf{z}_i^{(2)} \neq \mathbf{z}_j^{(2)}$.

Lemma H.3. Let $\mathbf{z}_1^{(1)}, \dots, \mathbf{z}_n^{(1)}$ be n distinct binary vectors of dimension $d^{(1)}$, and let $\mathbf{b}_1, \dots, \mathbf{b}_m$ be m vectors in $d^{(1)}$ dimensions, with i.i.d. Bernoulli(0.5) entries. Further, let $\mathbf{z}_1^{(2)}, \dots, \mathbf{z}_n^{(2)}$ be as defined in (H.1). If $m = 3 \log \frac{n}{\epsilon'}$, then with probability $1 - \epsilon'$,

$$\forall i \neq j : \mathbf{z}_i^{(2)} \neq \mathbf{z}_j^{(2)}.$$

The lemma above is a special case of the Gilbert-Varshamov bound (for example, see (Guruswami et al., 2012)). A simple proof of the lemma is provided in subsection H.1.1.3 for completeness.

Hence, we will be able to compress the $d^{(1)}$ -dimensional vectors down to m -dimensions. In particular this means we will be able to compress down from $\mathcal{O}(\frac{\log n}{\delta})$ to $\mathcal{O}(\log n)$ dimensions. However, note that the key in the operation above is the XOR operator, so we will need to use threshold activation to simulate the XOR operation. The next lemma says that we can transform $\mathbf{z}_i^{(1)}$ to $\mathbf{z}_i^{(2)}$ using a threshold network containing $\mathcal{O}(\frac{\log^2 n}{\delta})$ neurons and $\mathcal{O}(\frac{\log^2 n}{\delta})$ weights.

Lemma H.4. Let $\mathbf{z}_i^{(2)}$ be as defined in (H.1), for all $i \in [n]$. Then, we can construct a threshold network with $3d^{(1)}m$ neurons and $9d^{(1)}m$ weights and biases such that it outputs $\mathbf{z}_i^{(1)}$ when the input to it is $\mathbf{z}_i^{(2)}$, for all $i \in [n]$.

Hence, when the input to the overall network is x_i , the network constructed up till now will output $z_i^{(2)}$. Similar to the first layer, let $d^{(2)} = m$ denote the dimension of $z_i^{(2)}$. Note that this step uses $\mathcal{O}(\frac{\log^2 n}{\delta})$ neurons and $\mathcal{O}(\frac{\log^2 n}{\delta})$ weights.

Step 3: Partitioning the dataset into $\tilde{\mathcal{O}}(\sqrt{n})$ subsets of size at most $\sqrt{n}$ each. In this step, we partition $\{z_1^{(2)}, \dots, z_n^{(2)}\}$ into $\mathcal{O}(\sqrt{n} \log n)$ subsets of size at most $\sqrt{n}$ each. The reason why we do this will be apparent in **Step 4**. The key property that we want these subsets to have is that each subset should have a unique prefix such that all the $z_i^{(2)}$ that belong to one subset share that prefix. This will help in detecting which subset a particular $z_i^{(2)}$ belongs to, using threshold networks. Hence, first we will show that such prefixes exist.

Lemma H.5. *Given n distinct binary vectors $z_1^{(2)}, \dots, z_n^{(2)}$ of length $d^{(2)} = c \log_2 n$ each (for some constant $c \geq 1$), we can partition them into $\mathcal{O}(\sqrt{n} \log n)$ subsets of size at most $\sqrt{n}$ each, with the important property that each of these subsets will be characterized by unique prefixes, such that all of the elements of one subset will have the same prefix.*

Lemma H.5 gives us $K = \mathcal{O}(\sqrt{n} \log n)$ subsets which partition $\{z_1^{(2)}, \dots, z_n^{(2)}\}$, each with a unique prefix. Let the subsets be $S_1, \dots, S_K$. Let the binary prefix for S_i be b_i , which a length $1 \leq l \leq d^{(2)}$. Then the membership of a vector z can easily be detected by the neuron $s_i : \{0, 1\}^{d^{(2)}} \rightarrow \{0, 1\}$ given by $s_i(z) = \sigma(\sum_{j=1}^l (2b_{i,j} - 1)z_j - t)$, where t is the number of 1's in b_i . Hence, we can have K neurons, one for each subset, such that $s_i(z) = 1$ if and only if $z \in S_i$. Let $vs.(z)$ represent the vector formed after concatenating the outputs of all these neurons. Along with these K neurons, we will also have $d^{(2)}$ other neurons that simply copy $z^{(2)}$ over to the next layer. These $d^{(2)}$ neurons can simply have the form $f_i(z) = \sigma(2z_i - 1)$, for $i = 1, \dots, d^{(2)}$. Overall, if $z_i^{(2)} \in S_j$, then this layer outputs

$$\begin{bmatrix} vs.(z_i^{(2)}) \\ z_i^{(2)} \end{bmatrix}, \quad (\text{H.2})$$

as shown in the middle section of Figure 12.1. Thus, for **Step 3**, we have one layer which consists of K subset indicator neurons, along with $d^{(2)}$ neurons that simply copy z over to the output. Overall, in this step we used one layer with $K + d^{(2)}$ neuron and $(K + 1)d^{(2)}$ weights.

Step 4: The reason we divided the vectors into subsets and created the indicator neurons is so that the next layer can project the $d^{(2)} = \mathcal{O}(\log n)$ dimensional vectors $z_i^{(2)}$ to $d^{(3)} = \mathcal{O}(\sqrt{n} \log^2 n)$ dimension vectors using the subset indicators. Let $z_i^{(3)}$ be the projection of $z_i^{(2)}$. The way we project will ensure that if we take one vector from each of the K subsets,

then these vectors would be linearly independent. This linear independence of K vectors would help in memorizing K samples with just one neuron in the second-to-last layer, which means that we can memorize all the n samples with $\sqrt{n}$ neurons in the second-to-last layer. We explain all of this in detail now.

Step 4(a): Projecting the vectors up to dimension $\mathcal{O}(\sqrt{n} \log^2 n)$. In this layer, we will have $K(d^{(2)} + 1)$ neurons partitioned into groups of size $d^{(2)} + 1$ each. As explained in the main paper, the goal would be carry out the transformation shown in Figure 12.3.

To see how this happens, recall that the previous layer outputs (H.2). Then, the j -th neuron of the i -th group in this layer would have the form $g_{i,j} = \sigma(s_i - 1)$ for $j = 1$ and $g_{i,j} = \sigma_{i,j}(s_i + z_j^{(2)} - 2)$, for $j = 2, \dots, d^{(2)} + 1$. It can be verified that this implements the transformation shown in Figure 12.3.

This layer will have $K(d^{(2)} + 1)$ neurons and $\leq 3K(d^{(2)} + 1)$ weights and biases. Further, looking at neurons, note that the weights and bias are integers and less than 2.

Step 4(b): Memorizing using $\sqrt{n}$ neurons. In this layer, we have $\sqrt{n}$ neurons. The i -th neuron memorizes the i -th samples from each of the K subsets. Let $\mathbf{z}_{i,j}^{(2)}$ denote the i -th vector in S_j . The i -th neuron will have the weight vector given by

$$\mathbf{w}_i^{(3)} = \begin{bmatrix} -t_{i,1} + y_{i,1} - 1 \\ 2\mathbf{z}_{i,1}^{(2)} - \mathbf{1} \\ -t_{i,2} + y_{i,2} - 1 \\ 2\mathbf{z}_{i,2}^{(2)} - \mathbf{1} \\ \vdots \\ -t_{i,d^{(2)}} + y_{i,d^{(2)}} - 1 \\ 2\mathbf{z}_{i,d^{(2)}}^{(2)} - \mathbf{1} \end{bmatrix},$$

where $t_{i,j}$ is the number of 1's in $\mathbf{z}_{i,j}^{(2)}$, and $y_{i,j}$ is the label of sample $\mathbf{x}_{i,j}$ from the dataset that corresponds to $\mathbf{z}_{i,j}^{(2)}$. The biases of these neurons are 0.

To see how this works, consider the vector $\mathbf{z}_{i,j}^{(2)}$. This is the i -th vector in S_j . Then, the corresponding $\mathbf{z}_{i,j}^{(3)}$ would have support only in the j -th chunk of length $d^{(2)} + 1$. Then,

$$\begin{aligned} (\mathbf{w}_i^{(3)})^\top \mathbf{z}_{i,j}^{(3)} &= -t_{i,j} + y_{i,j} - 1 + (2\mathbf{z}_{i,j}^{(2)} - \mathbf{1})^\top \mathbf{z}_{i,j}^{(2)} \\ &= -t_{i,j} + y_{i,j} - 1 + 2t_{i,j} - t_{i,j} \\ &= y_{i,j} - 1. \end{aligned}$$

Hence, $\sigma((\mathbf{w}_i^{(3)})^\top \mathbf{z}_{i,j}^{(3)}) = y_{i,j}$. Further, for any $k \neq i$,

$$(\mathbf{w}_i^{(3)})^\top \mathbf{z}_{k,j}^{(3)} = -t_{i,j} + y_{i,j} - 1 + (2\mathbf{z}_{i,j}^{(2)} - \mathbf{1})^\top \mathbf{z}_{k,j}^{(2)}.$$

Note that $(2\mathbf{z}_{i,j}^{(2)} - \mathbf{1})^\top \mathbf{z}_{k,j}^{(2)} \leq t_{i,j} - 1$ if $\mathbf{z}_{i,j}^{(2)} \neq \mathbf{z}_{k,j}^{(2)}$. Hence,

$$(\mathbf{w}_i^{(3)})^\top \mathbf{z}_{k,j}^{(3)} \leq y_{i,j} - 2.$$

Hence, $\sigma((\mathbf{w}_i^{(3)})^\top \mathbf{z}_{k,j}^{(3)}) = 0$.

Thus we see that in this layer, the i -th neuron memorizes the i -th samples from each subset $\{S_1, \dots, S_K\}$, and for any other sample, it outputs 0. Since there are at most $\sqrt{n}$ samples in each subset, we need $\sqrt{n}$ neurons in this layer.

Let $[q_1, \dots, q_{\sqrt{n}}]$ be the concatenated output of this layer. Then, the last layer is a single neuron, $\sigma(2q_1 + \dots + 2q_{\sqrt{n}} - 1)$, which essentially just sums up the outputs of the previous layer. Since each neuron of the previous layer memorized a subset of the samples, this neuron is able to output the correct label for the entire dataset.

□

H.1.1 Proof of helper lemmas for Theorem 12.8

H.1.1.1 Proof of Lemma H.1

Proof. Consider any two samples from the dataset, $\mathbf{x}_i$ and $\mathbf{x}_j$. Let $\mathbf{u}$ and $\mathbf{v}$ be two orthonormal vectors that form a basis of the 2-dimensional space spanned by $\mathbf{x}_i$ and $\mathbf{x}_j$. Then, any point in the subspace can be written as $z_1\mathbf{u} + z_2\mathbf{v}$, for some scalars z_1 and z_2 . Let $\mathbf{x}_i = z_{i,1}\mathbf{u} + z_{i,2}\mathbf{v}$ and $\mathbf{x}_j = z_{j,1}\mathbf{u} + z_{j,2}\mathbf{v}$.

As per our construction of the hyperplanes, let $\mathbf{w}^\top \mathbf{x} = 0$ be any one of the m hyperplanes. Then, we want to find the probability that $(\mathbf{w}^\top \mathbf{x}_i)(\mathbf{w}^\top \mathbf{x}_j) < 0$. For this, it would be sufficient that $\mathbf{w}^\top \mathbf{x}_i > 0$ and $\mathbf{w}^\top \mathbf{x}_j < 0$. These are equivalent to $z_{i,1}\mathbf{w}^\top \mathbf{u} + z_{i,2}\mathbf{w}^\top \mathbf{v} > 0$ and $z_{j,1}\mathbf{w}^\top \mathbf{u} + z_{j,2}\mathbf{w}^\top \mathbf{v} < 0$. Note that since $\mathbf{w} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_d)$, we have that $\mathbf{w}^\top \mathbf{u} \sim \mathcal{N}(0, 1)$ and $\mathbf{w}^\top \mathbf{v} \sim \mathcal{N}(0, 1)$, and are independent. For convenience, define the random variables $g_1 := \mathbf{w}^\top \mathbf{u}$, $g_2 := \mathbf{w}^\top \mathbf{v}$, and $\mathbf{g} := [g_1 \ g_2]^\top$. Also, define $\mathbf{z}_i = [z_{i,1} \ z_{i,2}]^\top$ and $\mathbf{z}_j = [z_{j,1} \ z_{j,2}]^\top$. Then, we want to bound the probability of event

$$\{\mathbf{g}^\top \mathbf{z}_i > 0 \text{ and } \mathbf{g}^\top \mathbf{z}_j < 0\}. \quad (\text{H.3})$$

Note that $\mathbf{g}$, $\mathbf{z}_i$, and $\mathbf{z}_j$ are all in 2 dimensions, the angle between $\mathbf{z}_i$ and $\mathbf{z}_j$ is at least δ . Thus, for (H.3) to be true, $\mathbf{g}$ should lie in a cone of angle δ (see Figure 12.2). Noting that

g is rotationally symmetric, we get that the probability of (H.3) is exactly $\frac{\delta}{2\pi}$. Hence, the probability that the hyperplane does not separate z_i and z_j , is less than $1 - \frac{\delta}{2\pi}$. Since we have m independently sampled hyperplanes, the probability that none of them separates z_i and z_j is less than $(1 - \frac{\delta}{2\pi})^m$. This is the probability that one of the pairs x_i and x_j does not get separated. Since there are a total of $\binom{n}{2}$ pairs of points x_i and x_j , then using the union bound, the probability that there exists one pair of points such that the m hyperplanes do not separate them, is less than $\binom{n}{2}(1 - \frac{\delta}{2\pi})^m$. We want this probability to be less than ϵ , that is,

$$\binom{n}{2} \left(1 - \frac{\delta}{2\pi}\right)^m \leq \epsilon.$$

Note that $\binom{n}{2} \leq n^2$ and $\left(1 - \frac{\delta}{2\pi}\right)^m \leq e^{-\frac{\delta}{2\pi}m}$. Hence, it is sufficient for the following to hold

$$n^2 e^{-\frac{\delta}{2\pi}m} \leq \epsilon.$$

This gives that $m \geq \frac{2\pi}{\delta} \log\left(\frac{n^2}{\epsilon}\right)$ is sufficient for the lemma to be true. $\square$

H.1.1.2 Proof of Lemma H.2

This proof is similar to the proof of Lemma H.1.

Proof. Consider any two samples from the dataset, x_i and x_j . Let u and v be two orthonormal vectors that form a basis of the 2-dimensional space spanned by x_i and x_j , such that $v \perp (x_i - x_j)$. If x_i and x_j do not span a 2-dimensional space, then take u to be in the direction of x_i , and v can be in any other orthogonal direction. Then, any point in the subspace can be written as $z_1 u + z_2 v$, for some scalars z_1 and z_2 . Let $x_i = z_{i,1} u + z_{i,2} v$ and $x_j = z_{j,1} u + z_{j,2} v$. Note that because $v \perp (x_i - x_j)$, we have that $z_{i,2} = z_{j,2}$, and also $|z_{i,1} - z_{j,1}| = \|x_i - x_j\| \geq \delta$.

As per our construction of the hyperplanes, let $w^\top x + b = 0$ be any one of the m hyperplanes. We are interested in the probability that $(w^\top x_i + b_i)(w^\top x_j + b_j) < 0$, or equivalently

$$(z_{i,1} w^\top u + z_{i,2} w^\top v + b)(z_{j,1} w^\top u + z_{j,2} w^\top v + b) < 0. \quad (\text{H.4})$$

Note that since $w \sim \mathcal{N}(0, I_d)$, we have that $w^\top u \sim \mathcal{N}(0, 1)$ and $w^\top v \sim \mathcal{N}(0, 1)$, and are independent. For convenience, define the random variables $g_1 := w^\top u$, and $g_2 := w^\top v$. Let (s, t) be the coordinates in the 2 dimensional space spanned by u and v . Then, note

from (H.4) that we are interested in the behaviour of the hyperplane

$$g_1 s + g_2 t + b = 0, \quad (\text{H.5})$$

where g_1, g_2 are the coefficients and b is the bias. Note that this is just a line in 2-dimensions. In particular, (H.4) is still equivalent to the event that this line passes in between $\mathbf{x}_i$ and $\mathbf{x}_j$ in this 2-dimensional space.

Looking at the line (H.5), its slope is $-g_1/g_2$ and its (signed) distance from the origin is $b/\sqrt{g_1^2 + g_2^2}$. Next, note that if we consider the two dimensional Gaussian vector $[g_2 \ g_1]^\top \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_2)$, then thinking in terms of polar coordinates, $\sqrt{g_1^2 + g_2^2}$ is its norm and $-g_1/g_2$ is its direction. By the spherical symmetry of Gaussians, we know that the norm and its direction are independent. This implies that $b/\sqrt{g_1^2 + g_2^2}$ is independent of $-g_1/g_2$, that is the distance of the line from the origin is independent of its slope. Note that $r := \sqrt{2}b/\sqrt{g_1^2 + g_2^2}$ is the (scaled) ratio of a Gaussian random variable with the square-root of a χ^2 -random variable, and hence r has a Student's t -distribution with 2 degrees of freedom.

Fix the slope $\theta := -g_1/g_2$ of the line (H.5), then we have seen that the signed distance from the origin is the independent random variable $r/\sqrt{2}$. Then, for the line (H.5) to pass in between $\mathbf{x}_i, \mathbf{x}_j$, we will need to have that $\frac{r}{\sqrt{2}}$ should be between $\left(k \frac{z_{i,2} + \theta z_{i,1}}{\sqrt{1 + \theta^2}}\right)$ and $\left(k \frac{z_{j,2} + \theta z_{j,1}}{\sqrt{1 + \theta^2}}\right)$, where $k = -1$ or $k = 1$ depending on the sign of θ . These two exact values do not matter much. What is more important is that since $\|\mathbf{x}_i\| \leq 1$ and $\|\mathbf{x}_j\| \leq 1$, we know that these values are less than 1; and that

$$\begin{aligned} \left| \left(k \frac{z_{i,2} + \theta z_{i,1}}{\sqrt{1 + \theta^2}} \right) - \left(k \frac{z_{j,2} + \theta z_{j,1}}{\sqrt{1 + \theta^2}} \right) \right| &= \frac{|\theta|}{\sqrt{1 + \theta^2}} |z_{j,1} - z_{i,1}| \\ &\quad (\text{Since } k \in \{-1, 1\} \text{ and } z_{i,2} = z_{j,2}.) \\ &\geq \delta \frac{|\theta|}{\sqrt{1 + \theta^2}}. \end{aligned}$$

In short, once we fix the slope, θ , the probability that the line passes in between the two points is the same as that of a scaled Student's t -variable's value being in an interval of length at least $\delta \frac{|\theta|}{\sqrt{1 + \theta^2}}$. Note that the interval lies completely in $[-1, 1]$. Because in the interval $[-\sqrt{2}, \sqrt{2}]$, a Student's t -variable has its p.d.f. lower bounded by a universal constant C , we get that the probability that the line passes in between the two points is at least $2C\sqrt{2}\delta \frac{|\theta|}{\sqrt{1 + \theta^2}}$.

To compute a lower bound on the final probability, we need to unfix the slope θ and integrate the lower bound we computed above, over the distribution of the slope. Define the polar angle $\phi := \arctan(\theta) = \arctan(-g_1/g_2)$. Note that due to the spherical symmetry

of the Gaussian vector $[g_1 \ g_2]$, we will have that ϕ is uniformly distributed in $[0, 2\pi]$. Hence, the final probability is lower bounded by

$$\frac{1}{2\pi} \int_0^{2\pi} C\sqrt{2}\delta \frac{|\theta|}{\sqrt{1+\theta^2}} d\phi = \frac{1}{2\pi} C\sqrt{2}\delta \int_0^\pi |\sin(\phi)| d\phi = C'\delta,$$

for some universal constant C' .

What we have shown above, is that the probability that a hyperplane separates x_i and x_j is at least $C'\delta$. Hence, the probability that the hyperplane does not separate z_i and z_j , is less than $1 - C'\delta$. Since we have m independently sampled hyperplanes, the probability that none of them separate z_i and z_j , is less than $(1 - C'\delta)^m$. This is the probability that one of the pairs x_i and x_j does not get separated. Since there are a total of $\binom{n}{2}$ pairs of points x_i and x_j , then by using the union bound, the probability that there exists one pair of points such that the m hyperplanes does not separate them, is less than $\binom{n}{2}(1 - C'\delta)^m$. We want this probability to be less than ϵ , that is,

$$\binom{n}{2} (1 - C'\delta)^m \leq \epsilon.$$

Note that $\binom{n}{2} \leq n^2$ and $(1 - C'\delta)^m \leq e^{-C'\delta m}$. Hence, it is sufficient for the following to hold

$$n^2 e^{-C'\delta m} \leq \epsilon.$$

This gives that $m \geq \frac{1}{C'\delta} \log\left(\frac{n^2}{\epsilon}\right)$ is sufficient for the lemma to be true. □

H.1.1.3 Proof of Lemma H.3

Proof. Consider two binary vectors z_i and z_j that differ in at least one bit. Without loss of generality, assume that the first bit is one of the bits that they differ in. Let $z_{i,k}$ be the k -th element of z_i and similarly let $z_{j,k}$ be the k -th element of z_j . We can further assume without loss of generality that $z_{i,1} = 0$ and $z_{j,1} = 1$. Then, for any Bernoulli(0.5) vector $\mathbf{b}$, we have

$$\begin{aligned} \langle z_i, \mathbf{b} \rangle_\oplus &= (0 \cdot \mathbf{b}_1) \oplus (z_{i,2} \cdot \mathbf{b}_2) \oplus \cdots \oplus (z_{i,d'} \cdot \mathbf{b}_{d'}), \\ \langle z_j, \mathbf{b} \rangle_\oplus &= (1 \cdot \mathbf{b}_1) \oplus (z_{j,2} \cdot \mathbf{b}_2) \oplus \cdots \oplus (z_{j,d'} \cdot \mathbf{b}_{d'}), \end{aligned}$$

where $\mathbf{b}_k$ is the k -th element of $\mathbf{b}$. There can be two cases:

- $(z_{i,2} \cdot \mathbf{b}_2) \oplus \cdots \oplus (z_{i,d'} \cdot \mathbf{b}_{d'}) = (z_{j,2} \cdot \mathbf{b}_2) \oplus \cdots \oplus (z_{j,d'} \cdot \mathbf{b}_{d'})$. Say the probability of this is p . Note that $\mathbf{b}_1$ is independent of every other $\mathbf{b}_i$. With probability 0.5, $\mathbf{b}_1 = 1$. In this case,

$$\begin{aligned}
\langle \mathbf{z}_i, \mathbf{b} \rangle_{\oplus} &= (0 \cdot 1) \oplus (z_{i,2} \cdot \mathbf{b}_2) \oplus \cdots \oplus (z_{i,d'} \cdot \mathbf{b}_{d'}) \\
&= 0 \oplus (z_{i,2} \cdot \mathbf{b}_2) \oplus \cdots \oplus (z_{i,d'} \cdot \mathbf{b}_{d'}) \\
&= (z_{i,2} \cdot \mathbf{b}_2) \oplus \cdots \oplus (z_{i,d'} \cdot \mathbf{b}_{d'}) \\
&= \neg(\neg((z_{i,2} \cdot \mathbf{b}_2) \oplus \cdots \oplus (z_{i,d'} \cdot \mathbf{b}_{d'}))) \\
&\quad \text{(Applying two NOT operations does not affect the value.)} \\
&= \neg(1 \oplus (z_{i,2} \cdot \mathbf{b}_2) \oplus \cdots \oplus (z_{i,d'} \cdot \mathbf{b}_{d'})) \\
&\quad \text{(XOR with 1 is equivalent to the NOT operation.)} \\
&= \neg((1 \cdot 1) \oplus (z_{i,2} \cdot \mathbf{b}_2) \oplus \cdots \oplus (z_{i,d'} \cdot \mathbf{b}_{d'})) \\
&= \neg\langle \mathbf{z}_j, \mathbf{b} \rangle_{\oplus}.
\end{aligned}$$

Hence, in this case, with probability at least 0.5, $\langle \mathbf{z}_i, \mathbf{b} \rangle_{\oplus} \neq \langle \mathbf{z}_j, \mathbf{b} \rangle_{\oplus}$.

- $(z_{i,2} \cdot \mathbf{b}_2) \oplus \cdots \oplus (z_{i,d'} \cdot \mathbf{b}_{d'}) = \neg((z_{j,2} \cdot \mathbf{b}_2) \oplus \cdots \oplus (z_{j,d'} \cdot \mathbf{b}_{d'}))$. The probability of this is $1 - p$. Note that $\mathbf{b}_1$ is independent of every other $\mathbf{b}_i$. With probability 0.5, $\mathbf{b}_1 = 0$. In this case,

$$\begin{aligned}
\langle \mathbf{z}_i, \mathbf{b} \rangle_{\oplus} &= (0 \cdot 0) \oplus (z_{i,2} \cdot \mathbf{b}_2) \oplus \cdots \oplus (z_{i,d'} \cdot \mathbf{b}_{d'}) \\
&= 0 \oplus (z_{i,2} \cdot \mathbf{b}_2) \oplus \cdots \oplus (z_{i,d'} \cdot \mathbf{b}_{d'}) \\
&= (z_{i,2} \cdot \mathbf{b}_2) \oplus \cdots \oplus (z_{i,d'} \cdot \mathbf{b}_{d'}) \\
&= \neg((z_{j,2} \cdot \mathbf{b}_2) \oplus \cdots \oplus (z_{j,d'} \cdot \mathbf{b}_{d'})) \\
&= \neg((1 \cdot 0) \oplus (z_{j,2} \cdot \mathbf{b}_2) \oplus \cdots \oplus (z_{j,d'} \cdot \mathbf{b}_{d'})) \\
&= \neg\langle \mathbf{z}_j, \mathbf{b} \rangle_{\oplus}
\end{aligned}$$

Hence, in this case as well, with probability at least 0.5, $\langle \mathbf{z}_i, \mathbf{b} \rangle_{\oplus} \neq \langle \mathbf{z}_j, \mathbf{b} \rangle_{\oplus}$.

Hence overall, with probability at least $p(0.5) + (1 - p)(0.5) = 0.5$, we have that $\langle \mathbf{z}_i, \mathbf{b} \rangle_{\oplus} \neq \langle \mathbf{z}_j, \mathbf{b} \rangle_{\oplus}$. If we take m i.i.d. Bernoulli(0.5) vectors $\mathbf{b}_1, \dots, \mathbf{b}_m$, then the probability that

$$\forall k \in [m], \langle \mathbf{z}_i, \mathbf{b}_k \rangle_{\oplus} = \langle \mathbf{z}_j, \mathbf{b}_k \rangle_{\oplus},$$

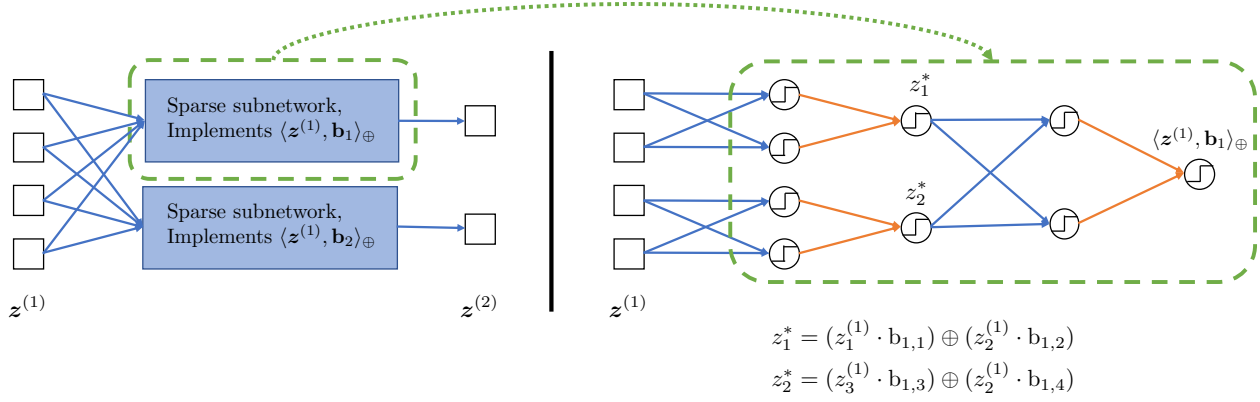


Figure H.1: **Left:** We can compute each of $\langle z^{(1)}, b_k \rangle$ using subnetworks shown in blue, that we can put in parallel to output $z^{(2)}$. **Right:** A subnetwork that implements $\langle z^{(1)}, b_1 \rangle$. This, in turn can be broken down as $\langle z^{(1)}, b_1 \rangle_{\oplus} = (z_1^{(1)} \cdot b_{1,1}) \oplus (z_2^{(1)} \cdot b_{1,2}) \oplus \dots \oplus (z_{d^{(1)}}^{(1)} \cdot b_{1,d^{(1)}})$. Then, the figure on the right shows how the network computes $\langle z^{(1)}, b_1 \rangle$ using a binary tree structure (see the orange lines in the figure). In the figure, we take $d^{(1)} = 4$. The network first computes $z_1^* = (z_1^{(1)} \cdot b_{1,1}) \oplus (z_2^{(1)} \cdot b_{1,2})$ and $z_2^* = (z_3^{(1)} \cdot b_{1,3}) \oplus (z_4^{(1)} \cdot b_{1,4})$. Then, it computes $z_1^* \oplus z_2^*$, as shown in the figure.

is less than $(0.5)^m$. Equivalently,

$$\Pr(z_i^{(2)} = z_j^{(2)}) \leq 2^{-m}.$$

To get the bound required in the lemma statement, we simply take a union bound over all the $\binom{n}{2}$ pairs $(z_i^{(2)}, z_j^{(2)})$. Concretely, we want $\Pr(\exists i, j : z_i^{(2)} = z_j^{(2)}) \leq \epsilon$. Hence, we take a union bound:

$$\begin{aligned} \Pr(\exists i, j : z_i^{(2)} = z_j^{(2)}) &\leq \binom{n}{2} \Pr(z_i^{(2)} = z_j^{(2)}) \\ &\leq \binom{n}{2} 2^{-m}. \end{aligned}$$

It can be easily verified that $m \geq 3 \log \frac{n}{\epsilon}$ suffices for $\binom{n}{2} 2^{-m} \leq \epsilon$. This completes the proof. $\square$

H.1.1.4 Proof of Lemma H.4

Proof. We will show that we can compute $\langle z_i^{(1)}, b_k \rangle_{\oplus}$ with a network of $3(d^{(1)} - 1)$ neurons and $9(d^{(1)} - 1)$ weights. Then, we can put m such networks in parallel (see Figure H.1, Left)

to output

$$\mathbf{z}_i^{(2)} = \begin{bmatrix} \langle \mathbf{b}_1, \mathbf{z}_i \rangle_{\oplus} \\ \vdots \\ \langle \mathbf{b}_m, \mathbf{z}_i \rangle_{\oplus} \end{bmatrix}.$$

Hence, all we need to do is to prove that we can compute $\langle \mathbf{z}_i^{(1)}, \mathbf{b}_k \rangle_{\oplus}$ with a network of $3(d^{(1)} - 1)$ neurons and $9(d^{(1)} - 1)$ weights. We will do this in a hierarchical fashion as shown in Figure H.1, Right. We begin by expanding $\langle \mathbf{z}_i^{(1)}, \mathbf{b}_k \rangle_{\oplus}$:

$$\langle \mathbf{z}_i^{(1)}, \mathbf{b}_k \rangle_{\oplus} = (z_{i,1}^{(1)} \cdot \mathbf{b}_{k,1}) \oplus (z_{i,2}^{(1)} \cdot \mathbf{b}_{k,2}) \oplus \cdots \oplus (z_{i,d^{(1)}}^{(1)} \cdot \mathbf{b}_{k,d^{(1)}}).$$

Note that

$$\begin{aligned} & (z_{i,1}^{(1)} \cdot \mathbf{b}_{k,1}) \oplus (z_{i,2}^{(1)} \cdot \mathbf{b}_{k,2}) \\ &= \sigma(\sigma((z_{i,1}^{(1)} \cdot \mathbf{b}_{k,1}) - (z_{i,2}^{(1)} \cdot \mathbf{b}_{k,2}) - 1) + \sigma(-(z_{i,1}^{(1)} \cdot \mathbf{b}_{k,1}) + (z_{i,2}^{(1)} \cdot \mathbf{b}_{k,2}) - 1))), \end{aligned}$$

which uses 3 neurons, each with 2 weights and a bias. Similarly, we can compute $(z_{i,3}^{(1)} \cdot \mathbf{b}_{k,3}) \oplus (z_{i,4}^{(1)} \cdot \mathbf{b}_{k,4})$, $(z_{i,5}^{(1)} \cdot \mathbf{b}_{k,5}) \oplus (z_{i,6}^{(1)} \cdot \mathbf{b}_{k,6})$, ..., $(z_{i,d^{(1)}-1}^{(1)} \cdot \mathbf{b}_{k,d^{(1)}-1}) \oplus (z_{i,d^{(1)}}^{(1)} \cdot \mathbf{b}_{k,d^{(1)}})$. Once we have computed these, we can compute XOR's of four terms similarly, to get terms like $(z_{i,1}^{(1)} \cdot \mathbf{b}_{k,1}) \oplus (z_{i,2}^{(1)} \cdot \mathbf{b}_{k,2}) \oplus (z_{i,3}^{(1)} \cdot \mathbf{b}_{k,3}) \oplus (z_{i,4}^{(1)} \cdot \mathbf{b}_{k,4})$ (see Figure H.1, Right). The tree will consist of $d^{(1)} - 1$ XOR operations, each represented by a node. For each of these operations, we saw above that we need 3 neurons, each with 2 weights and a bias. This gives that we need $3(d^{(1)} - 1)$ neurons and $9(d^{(1)} - 1)$ weights and biases to compute $\langle \mathbf{z}_i^{(1)}, \mathbf{b}_k \rangle_{\oplus}$. □

H.1.1.5 Proof of Lemma H.5

Proof. Consider a binary tree of depth $d^{(2)} + 1$, where the left child edge of each node is labeled 0, and the right one is labeled 1. There will be $2^{d^{(2)}}$ leaves, each representing one unique binary vector of length $d^{(2)}$. The vector associated with a leaf would be the same as the sequence of edges of the path from the root to that leaf. Hence, $\{\mathbf{z}_1^{(2)}, \dots, \mathbf{z}_n^{(2)}\}$ will form a subset of the $2^{d^{(2)}}$ leaves. Remove all other leaves from the tree, so that we only have $\{\mathbf{z}_1^{(2)}, \dots, \mathbf{z}_n^{(2)}\}$ as the leaves in our tree. Hence, now our tree only has n leaves. Also remove all the nodes that are not the ancestor of any leaves.

The way we select subsets is simple: Each node is a potential subset. Each node

represents the subset of $\{z_1^{(2)}, \dots, z_n^{(2)}\}$ that belong in the subtree at that node. Each node is also associated with a unique prefix: the sequence of edges of the path from the root to that node. Denote by $\mathcal{S}$ the set of nodes (or equivalently subsets) that we select. We include a node in $\mathcal{S}$, if

1. the number of leaves in the subtree at that node is at most $\sqrt{n}$,
2. none of its ancestor nodes satisfy condition 1.

Condition 1 will ensure that the subsets formed in this way will have size at most $\sqrt{n}$. Condition 2 above ensures that the subsets do not overlap and consequently the prefixes of the selected nodes are unique. Hence, all that is left to do is prove that $|\mathcal{S}| = \mathcal{O}(\sqrt{n} \log n)$.

Define the height of a node as $(d^{(2)} - \text{its depth})$. Thus, the root node has height $d^{(2)}$ and the leaves have height 0. We will prove that for any node at height h , if any of its descendant nodes is chosen, then

$$\frac{\# \text{ leaves in the subtree at that node}}{\# \text{ descendant nodes included in } \mathcal{S}} \geq \frac{\sqrt{n}}{h}. \quad (\text{H.6})$$

At root node, $h = d^{(2)} = c \log n$, and $\# \text{ leaves} = n$. Then, rearranging the inequality (H.6), we get that at the root node the number of descendant nodes included in $\mathcal{S}$ are less than $\sqrt{n}(c \log n)$. Thus, the total number of subsets is $\mathcal{O}(\sqrt{n} \log n)$. Hence, all we need to do now, is to prove (H.6). For this, we will use induction on h .

The smallest h for which the inequality is applicable, will be $h = \lceil \log_2 \sqrt{n} \rceil + 1$. This is because the first height at which a node can be selected is $\lceil \log_2 \sqrt{n} \rceil$. Note that no descendants of a node at height $h = \lceil \log_2 \sqrt{n} \rceil + 1$, other than its children, can be included in $\mathcal{S}$. Further, since one of its children is included in $\mathcal{S}$, this node itself is not included in $\mathcal{S}$. Hence, the number of leaves in the subtree at this node is at least $\sqrt{n} + 1$. Hence for this node,

$$\frac{\# \text{ leaves in the subtree formed at that node}}{\# \text{ descendant nodes in } \mathcal{S}} \geq \frac{\sqrt{n} + 1}{2} \geq \frac{\sqrt{n}}{h}.$$

Thus, the base case is proved. We move on to the inductive case now: $h > \lceil \log_2 \sqrt{n} \rceil + 1$. There can be the following cases for a node s at height h .

- s has only one child. This child cannot be in $\mathcal{S}$, since Condition 2 would be violated. Thus, it can only be the case that this child has further descendants that are in $\mathcal{S}$. Then, by the induction hypothesis, (H.6) is satisfied for the child, and hence it is also satisfied for s .

- Both of its children have descendants that are in $\mathcal{S}$. Let l_l be the (# leaves in the subtree formed at the left child of s) and d_l be the (# descendant nodes of the left child in $\mathcal{S}$). Similarly, we define l_r and d_r for the right child. Then, we know by the induction hypothesis that

$$\frac{l_l}{d_l} \geq \frac{\sqrt{n}}{h-1}, \frac{l_r}{d_r} \geq \frac{\sqrt{n}}{h-1}.$$

Hence, for the node s ,

$$\begin{aligned} \frac{\# \text{ leaves in the subtree formed at } s}{\# \text{ descendant nodes in } \mathcal{S}} &= \frac{l_l + l_r}{d_l + d_r} \\ &\geq \frac{\frac{\sqrt{n}}{h-1}(d_l + d_r)}{d_l + d_r} \\ &\geq \frac{\sqrt{n}}{h}. \end{aligned}$$

- One of its children is included in $\mathcal{S}$ and the other node has at least one of its descendants included in $\mathcal{S}$. Let the left child be the one that is included in $\mathcal{S}$. Let l_r be the (# leaves in the subtree formed at the right child of s) and d_r be the (# descendant nodes of the right child included in $\mathcal{S}$). Then, by inductive hypothesis,

$$\frac{l_r}{d_r} \geq \frac{\sqrt{n}}{h-1}.$$

We also know that since the right child wasn't included in $\mathcal{S}$, $l_r > \sqrt{n}$. Note that the number of leaves under s is at least l_r and the number of its descendants that are in $\mathcal{S}$ is $d_r + 1$, including the left child. Then,

$$\begin{aligned} \frac{\# \text{ leaves in the subtree formed at } s}{\# \text{ descendant nodes in } \mathcal{S}} &\geq \frac{l_r}{1 + d_r} \\ &= \frac{1}{\frac{1}{l_r} + \frac{d_r}{l_r}} \\ &\geq \frac{1}{\frac{1}{l_r} + \frac{h-1}{\sqrt{n}}} \\ &= \frac{\sqrt{n}}{\frac{\sqrt{n}}{l_r} + h-1} \\ &> \frac{\sqrt{n}}{h}. \end{aligned}$$

- Both the children of the node are in $\mathcal{S}$. In this case, since the node itself wasn't chosen, the number of leaves in the subtree is at least $\sqrt{n} + 1$, and the number of descendant nodes in $\mathcal{S}$ is 2. Hence,

$$\frac{\# \text{ leaves in the subtree formed by that node}}{\# \text{ descendant nodes in } \mathcal{S}} \geq \frac{\sqrt{n} + 1}{2} \geq \frac{\sqrt{n}}{h}.$$

This proves the inductive case and completes the proof of lemma. $\square$

H.2 Proof of Theorem 12.9 and Theorem 12.16

Proof. We want to construct an example where there are n points on the unit sphere $\mathcal{S}^{d-1}$, which are separated by a distance at least δ , such that the minimum number of hyperplanes needed for separating each pair is $\Omega(\log(n)/\delta)$ and for separating pairs with opposite labels is $\Omega(1/\delta)$.

Before we start the proof, let us define some notation. We denote by $\mathcal{S}^{d-1} := \{x : \|x\| = 1\}$, the unit sphere in d -dimensions centered at the origin. We can also have lower dimensional spheres in the same space. For example, we can have circles in a 3-dimensional space. We denote by $\mathcal{S}_c^{d-2}$, the $d - 1$ dimensional unit sphere, centered at c and which lies on the hyperplane orthogonal to the vector c . For example, in 3-dimensions, let $c = (1, 1, 1)$. Let $\mathcal{H}_c$ be the hyperplane that passes through c and is orthogonal to the vector connecting the origin to c . On this hyperplane, we can draw a unit circle, centered at $(1, 1, 1)$, which is what we denote by $\mathcal{S}_{(1,1,1)}^2$ in our notation. Further, $r\mathcal{S}^{d-1}$ and $r\mathcal{S}_c^{d-2}$ will denote the corresponding hyperspheres, but with radius r instead of 1. We denote the cluster centered at c by $\mathcal{Q}_c$. We will use $C_1, C_2, \dots$ to denote universal constants.

Now, we are ready to start the construction. We will create $\sqrt{n}$ clusters of points on the sphere, each of size $\sqrt{n}$.

First, the centers of the $\sqrt{n}$ clusters are sampled uniformly from $(\sqrt{1 - \delta})\mathcal{S}^{d-1}$. Then, in each cluster, $\sqrt{n}$ points are sampled i.i.d. from $\sqrt{\delta}\mathcal{S}_c^{d-2}$, where c is the center of the cluster. This way, each point has norm $\sqrt{(\sqrt{1 - \delta})^2 + (\sqrt{\delta})^2} = 1$, that is, each point indeed lies on $\mathcal{S}^d$. Of the $\sqrt{n}$ points in each cluster, we label half of them as 0 and the other half as 1.

Then, roughly speaking, the summary of the proof is as follows:

- Any hyperplane can only 'effectively' separate points within a cluster if it passes close to the center of the cluster. The distance would roughly need to be $\tilde{O}(\sqrt{\delta}/\sqrt{d})$.
- A hyperplane can be $\tilde{O}(\sqrt{\delta}/\sqrt{d})$ close to the centers of only $\tilde{O}(\sqrt{\delta})$ fraction of the clusters.

- Any cluster needs at least
 - $\Omega(\log_2 \sqrt{n}) = \Omega(\log n)$ hyperplanes to separate each pair of points within the cluster (for Theorem 12.16) or
 - at least 1 hyperplane to separate all the opposite label pairs (for Theorem 12.9).
- Finally, the core inequality that we use is that

$$\begin{aligned}
 & (\# \text{ hyperplanes needed}) \\
 & \quad \times (\# \text{ clusters a hyperplane can 'effectively' separate}) \\
 & \quad \geq (\# \text{ clusters}) \times (\# \text{ hyperplanes needed per cluster}). \tag{H.7}
 \end{aligned}$$

Substituting the bounds we got above into this inequality, we get that we need at least $\Omega(\log(n)/\sqrt{\delta})$ hyperplanes.

Now we describe the construction in detail. Note that we only need to prove the existence of *one* such construction. Thus, we will use the probabilistic method.

We define four events:

$\mathcal{E}_1$: Two points from different clusters have distance less than δ .

$\mathcal{E}_2$: The points within a cluster have distance less than δ .

$\mathcal{E}_3$: A hyperplane at distance $\Omega(\sqrt{\frac{\delta \log t}{d}})$ from the center of a cluster separates $\Omega(n/t)$ pairs of points in a cluster, for $t \geq 4$.

$\mathcal{E}_4$: A hyperplane passes within distance $O(\sqrt{\frac{\delta \log t}{d}})$ of the centers of $\Omega(\sqrt{\delta n \log t})$ clusters.

We will prove that the probability of each of these events is at most $1/5$. Hence, a union bound shows that there exists an event where none of these events happen. We will prove that such an event gives the required lower bounds.

Step 1: Bound the probability of $\mathcal{E}_1$. Let $c_1, \dots, c_{\sqrt{n}}$ be $\sqrt{n}$ cluster centers sampled uniformly from $(\sqrt{1-\delta})\mathcal{S}^{d-1}$, representing the centers of the clusters. Then there exists a universal constant C , such that, as long as $n < \left(\frac{C}{\sqrt{\delta}}\right)^{d-1}$, we show that the event that two points from different clusters have distance less than δ will have probability less than $1/5$.

To see how this is true, consider any two clusters, with centers c_i and c_j . Then, as long as $\|c_i - c_j\| \geq 2\sqrt{\delta} + \delta$, we get by triangle inequality that two points from these two clusters

are at least δ distance apart. Since $\sqrt{\delta} \geq \delta$, we know that if we ensure that all cluster centers are at least $3\sqrt{\delta}$ apart, then the above still holds.

Hence, we need to show that if we sample $\sqrt{n}$ points (cluster centers) on a sphere of radius $\sqrt{1-\delta}$, and $n < \frac{1}{5} \left(\frac{C}{\sqrt{\delta}}\right)^{d-1}$, then they are at least $3\sqrt{\delta}$ apart with probability at least $4/5$. For this, we need to show that the probability of any pair of points c_i and c_j being within $3\sqrt{\delta}$ distance is less than $\left(\frac{\sqrt{\delta}}{C}\right)^{d-1}$. Taking a union bound over all the $\binom{\sqrt{n}}{2}$ pairs and substituting $n < \left(\frac{C}{\sqrt{\delta}}\right)^{d-1}$ will give that the probability of any such pair existing is less than $1/5$.

Thus, all we need to do is to prove that the probability of any pair of points being within $3\sqrt{\delta}$ distance is less than $\left(\frac{\sqrt{\delta}}{C}\right)^{d-1}$. We prove it below.

Fix the point c_i . Then, the probability of any other cluster center c_j being within distance $3\sqrt{\delta}$ of c_i is the same as the ratio of the surface area of the cap of radius $3\sqrt{\delta}$ to the surface area of $\sqrt{1-\delta}S^{d-1}$. Consider packing the surface of $\sqrt{1-\delta}S^{d-1}$ with caps of radius $3\sqrt{\delta}$, and let $\mathcal{A}_{3\sqrt{\delta}}$ denote such a cap. Let the number of caps required for the packing be $\mathcal{P}'$. Then,

$$\begin{aligned} \text{Area}(3\sqrt{\delta} \text{ cap}) \times \mathcal{P}' &\leq \text{Area}(\sqrt{1-\delta}S^{d-1}) \\ \implies \frac{\text{Area}(3\sqrt{\delta} \text{ cap})}{\text{Area}(\sqrt{1-\delta}S^{d-1})} &\leq \frac{1}{\mathcal{P}'} \end{aligned}$$

Further, note that by rescaling $|\mathcal{P}'|$ is equal to $|\mathcal{P}_{3\sqrt{\delta}/\sqrt{1-\delta}}|$, where $\mathcal{P}_{\delta'}$ is the δ' packing of S^{d-1} . Finally, we use the following lemma (proved later in the appendix) to conclude that the probability of any pair of points being within $3\sqrt{\delta}$ distance is less than $\left(\frac{\sqrt{\delta}}{C}\right)^{d-1}$, for some universal constant C .

Lemma H.6. *Let $\mathcal{C}_\delta$ and $\mathcal{P}_\delta$ denote the covering and packing numbers of S^{d-1} , respectively. Then,*

$$\left(\frac{1}{4\delta}\right)^{d-1} \leq \mathcal{C}_\delta \leq \mathcal{P}_\delta \leq \left(\frac{2}{\delta}\right)^d.$$

Step 2: Bound the probability of $\mathcal{E}_2$. Consider any one cluster center c . Then, the sphere $\sqrt{\delta}S_c^{d-2}$ is just a $d-2$ dimensional sphere. We create this cluster by sampling points uniformly on S_c^{d-2} . This is similar to what we did in previous step and hence we can apply exactly the same steps as above to get that as long as $n < \left(\frac{C}{\sqrt{\delta}}\right)^{d-2}$, we have that the event that two points within some cluster have distance less than δ , will have probability less than $1/5$. We denote this cluster by $\mathcal{Q}_c$. Then, $\mathcal{Q}_c$ lies on the sphere $\sqrt{\delta}S_c^{d-2}$, which in turn lies on the hyperplane $\mathcal{H}_c$.

We need to create such clusters around each of the other cluster centers. A trick that we employ is that we use the same cluster $\mathcal{Q}_c$ and ‘paste’ it at every other cluster center, after rotating appropriately to make the cluster lie on the hyperplane orthogonal to the corresponding cluster center. This is helpful because we would not have to take a union bound of the probability calculation done above for every cluster. We formally describe the procedure below.

Let c' be another cluster center. Then, there will be a corresponding sphere $\sqrt{\delta}S_{c'}^{d-2}$, which lies on the hyperplane $\mathcal{H}_{c'}$. Let $T_{c'}$ be any linear transformation such that $T(\sqrt{\delta}S_c^{d-2}) = \sqrt{\delta}S_{c'}^{d-2}$. Then, we create the cluster $\mathcal{Q}_{c'}$ for the cluster center c' as $\mathcal{Q}_{c'} = T(\mathcal{Q}_c)$.

Step 3: Bound the probability of $\mathcal{E}_3$. Any hyperplane $\mathcal{H}_2$ (distinct from $\mathcal{H}_c$) that intersects $\mathcal{H}_c$ creates a $d - 1$ dimensional hyperplane, say $\mathcal{H}'_2$. We want to show that if $\mathcal{H}_2$ lies at distance $\Omega(\sqrt{\frac{\delta \log t}{d}})$ from a cluster center c , then the probability that it separates $\geq n/t$ pairs of points in the cluster is less than $1/5$. We will actually show that if $\mathcal{H}'_2$ lies at distance $\Omega(\sqrt{\frac{\delta \log t}{d}})$ from c , then the probability that it separates $\Omega(n/t)$ pairs of points in the cluster is less than $1/5$, which is a stronger statement because the distance of c from $\mathcal{H}_2$ is smaller than the distance of c from $\mathcal{H}'_2$. We do not consider hyperplanes parallel to $\mathcal{H}_c$ because such hyperplanes do not separate any points in $\mathcal{Q}_c$. We will work in the $d - 1$ dimensional subspace spanned by $\mathcal{H}_{c'}$ with c as the origin.

Take any one direction in this space. The probability measure of $\sqrt{\delta}S_c^{d-2}$ at distance $\geq 2\sqrt{\frac{\delta \log t}{d-1}}$ in this direction is at most $e^{-2 \log t}$ (Ball et al., 1997, Lemma 2.2). Thus, the probability that a point in the cluster lies at a distance more than $2\sqrt{\frac{\delta \log t}{d-1}}$ is less than t^{-2} . We can now apply Binomial tail probability bounds to get a probability bound on the event

$$E : \left\{ \begin{array}{l} \text{More than } \frac{1}{t} \text{ fraction of the } \sqrt{n} \text{ points lie at a distance more than } 2\sqrt{\frac{\delta \log t}{d-1}} \\ \text{in one fixed direction} \end{array} \right\}.$$

Let $D(\cdot \| \cdot)$ denote the KL-divergence. Then, the Chernoff bound on the tail probability of a Binomial random variable (Arratia and Gordon, 1989) gives the following bound

$$\begin{aligned} \Pr(E) &\leq e^{-\sqrt{n}D(\frac{1}{t} \| \frac{1}{t^2})} \\ &= \exp \left(-\sqrt{n} \left(\frac{1}{t} \log t + \left(1 - \frac{1}{t} \right) \log \frac{1 - \frac{1}{t}}{1 - \frac{1}{t^2}} \right) \right) \\ &= \exp \left(-\sqrt{n} \left(\frac{1}{t} \log t + \left(1 - \frac{1}{t} \right) \log \frac{t}{1+t} \right) \right) \end{aligned}$$

$$\begin{aligned}
&= \exp \left(-\sqrt{n} \left(\log t - \log(1+t) + \frac{1}{t} \log(1+t+t^2) \right) \right) \\
&= \exp \left(-\sqrt{n} \left(\log \left(\frac{t}{1+t} \right) + \frac{1}{t} \log(1+t) \right) \right) \\
&= \exp \left(-\sqrt{n} \left(\log \left(1 - \frac{1}{1+t} \right) + \frac{1}{t} \log(1+t) \right) \right) \\
&\leq \exp \left(-\sqrt{n} \left(-\frac{1}{2(1+t)} + \frac{1}{t} \log(1+t) \right) \right) \quad \left(\text{Since } \frac{1}{1+t} \leq 1 \right) \\
&\leq \exp \left(\frac{-\sqrt{n}}{2t} \log(1+t) \right) \quad \left(\text{Since } t \geq 2 \right) \\
&\leq \frac{1}{t^{\sqrt{n}/2t}}.
\end{aligned}$$

Recall that the above result is just for one fixed direction, whereas we want to ensure that no more than $1/t$ fraction of points lie at distance $\Omega(\sqrt{\delta \log t / (d-1)})$ in *any* direction. For this, we need an ϵ -net argument¹ over all the directions. Note that a direction is essentially just a point on the unit sphere and hence, we need to make an ϵ -net on S^{d-2} . We claim that for $\epsilon = \sqrt{\frac{\log t}{d-1}}$, if we prove that no more than $1/t$ fraction of points lie at distance $2\sqrt{\delta \log t / (d-1)}$ in any direction in the ϵ -net, then no more than $1/t$ fraction of points lie at distance $4\sqrt{\delta \log t / (d-1)}$ in any direction. We will prove this claim shortly. Recall that an ϵ -net is the same as a covering and hence we can use Lemma H.6 to get an upper bound on the size of the ϵ -net (Note that the radius of the sphere here is $\sqrt{\delta}$, so we need to rescale appropriately). Taking a union bound over the ϵ -net gives the probability

$$p := \left(4\sqrt{\frac{d-1}{\log t}} \right)^{d-1} \frac{1}{t^{\sqrt{n}/2t}}. \quad (\text{H.8})$$

Thus, any hyperplane that lies at distance more than $4\sqrt{\frac{\delta \log t}{d-1}}$ from the center of a cluster has less than $\sqrt{n}/t$ points on one side of it, with probability at least $1 - p$. Note that if a hyperplane has less than $\sqrt{n}/t$ points on one side, then it can only separate $\leq \frac{\sqrt{n}}{t}(\sqrt{n} - \frac{\sqrt{n}}{t}) \leq \frac{n}{t}$ pairs of points, which is what we want to prove in this step. Hence, we want to show that $p < 1/5$. From (H.8), we can see that this will be true as long as $\sqrt{n} \geq C_3 d t \log d$, where C_3 is a large enough universal constant.

To finish this step, we need to prove that an ϵ -net would ensure that in every direction, no more than $1/t$ fraction of points lie beyond $4\sqrt{\frac{\delta \log t}{d-1}}$. For any direction (or equivalently, a unit vector) $\mathbf{u}$, there is a direction $\mathbf{u}_e$ in our epsilon net such that $\|\mathbf{u} - \mathbf{u}_e\| \leq 2\sqrt{\frac{\log t}{d-1}}$. Consider the region of $\sqrt{\delta} S_c^{d-2}$ consisting of points that lie at distance $4\sqrt{\frac{\delta \log t}{d-1}}$ from the

¹An ϵ -net $\mathcal{E}$ of a set $\mathcal{B}$ is a set of points from $\mathcal{B}$ such that any point in $\mathcal{B}$ is at distance at most ϵ from some point in $\mathcal{E}$.

origin (*i.e.*, $\mathbf{c}$) in the direction of $\mathbf{u}$. Let $\mathbf{x}$ be any point in the region. Then,

$$\begin{aligned}
 \langle \mathbf{x}, \mathbf{u}_e \rangle &= \langle \mathbf{x}, \mathbf{u} \rangle + \langle \mathbf{x}, \mathbf{u}_e - \mathbf{u} \rangle \\
 &\geq 4\sqrt{\frac{\delta \log t}{d-1}} + \langle \mathbf{x}, \mathbf{u}_e - \mathbf{u} \rangle \\
 &\geq 4\sqrt{\frac{\delta \log t}{d-1}} - \|\mathbf{x}\| \|\mathbf{u}_e - \mathbf{u}\| \\
 &\geq 4\sqrt{\frac{\delta \log t}{d-1}} - 2\sqrt{\frac{\delta \log t}{d-1}} \\
 &= 2\sqrt{\frac{\delta \log t}{d-1}}.
 \end{aligned}$$

Hence, any point in such a region lies at distance $\geq 2\sqrt{\frac{\delta \log t}{d-1}}$ in the direction $\mathbf{u}_e$.

Step 4: Bound the probability of $\mathcal{E}_4$. This is similar to the previous step. We need to show that there are only $\mathcal{O}(\sqrt{\delta n \log t})$ cluster centers within distance $\mathcal{O}(\sqrt{\delta \log t/d})$ of any hyperplane.

Take any fixed hyperplane. Then, the probability measure of the sphere $\sqrt{1-\delta} \mathcal{S}^{d-1}$ within distance $12\sqrt{\delta \log t}/\sqrt{d}$ of the hyperplane is less than $C_5\sqrt{\delta \log t}$, for some universal constant C_5 . Using this, we get that the probability that any cluster center lies within distance $12\sqrt{\delta \log t}/\sqrt{d}$ of the hyperplane is less than $C_5\sqrt{\delta \log t}$. Similar to the previous step, we can now use Chernoff bounds to get the probability that more than $2C_5\sqrt{\delta \log t}$ fraction of cluster centers lie at a distance less than $12\sqrt{\delta \log(t)/d}$ is less than $e^{-C_5\sqrt{n\delta \log t}}$.

Recall that this is for a fixed hyperplane, whereas we want to give a probability bound for all hyperplanes. Hence, similar to the previous step, we take a union bound over an ϵ -net of hyperplanes. We claim that if $\epsilon = 4\sqrt{\delta \log(t)/d}$, then the ϵ -net argument will ensure that for any hyperplane, the probability that more than $2C_5\sqrt{\delta \log t}$ fraction of cluster centers lie at a distance less than $4\sqrt{\delta \log(t)/d}$ is less than $e^{-C_5\sqrt{n\delta \log t}}$. We will prove this shortly. We will see that the size of the net will be $\frac{1}{\epsilon}$ times the covering number of $\mathcal{S}^{d-1}$. Taking a union bound over the ϵ -net gives the probability

$$p' := \frac{1}{\epsilon} \left(C_6 \sqrt{\frac{d}{\delta \log t}} \right)^{d-1} e^{-C_5\sqrt{n\delta \log t}}.$$

p' will be less than $1/5$ when $\sqrt{n} > C_7 \frac{d}{\sqrt{\delta}} \log \frac{d}{\delta}$.

Next, we prove that if we create an ϵ -net of hyperplanes, then that would indeed be

sufficient to ensure that for any hyperplane, the probability that more than $2C_5\sqrt{\delta \log t}$ fraction of cluster centers lie at a distance less than $4\sqrt{\delta \log(t)/d}$ is less than $e^{-C_5\sqrt{n\delta \log t}}$. A hyperplane can be represented by $h(\mathbf{x}) = \mathbf{u}^\top \mathbf{x} + b$, where $\mathbf{u}$ is a unit vector. Similar to previous step, create an ϵ -net over the unit vectors $\mathbf{u}$. We also create an ϵ -net over b . However, since we are only interested in hyperplanes that intersect the sphere, we restrict $b \in [0, 1]$, and hence the epsilon net over b would be of size $1/\epsilon$. We know that there is a hyperplane $h_e(\mathbf{x}) = \mathbf{w}_e^\top \mathbf{x} + b_e$ in our ϵ -net such that $\|\mathbf{w}_e - \mathbf{w}\| \leq \epsilon$ and $|b - b_e| \leq \epsilon$. Consider the region of $\sqrt{1 - \delta}S^{d-1}$ within distance $4\sqrt{\delta \log(t)/d}$ of h . Hence, for any point $\mathbf{x}$ in the region,

$$\begin{aligned} \mathbf{w}_e^\top \mathbf{x} + b_e &= (\mathbf{w}^\top \mathbf{x} + b) + (\mathbf{w}_e - \mathbf{w})^\top \mathbf{x} + (b_e - b) \\ &\leq (\mathbf{w}^\top \mathbf{x} + b) + (\mathbf{w}_e - \mathbf{w})^\top \mathbf{x} + \epsilon \\ &\leq (\mathbf{w}^\top \mathbf{x} + b) + \|\mathbf{w}_e - \mathbf{w}\| \|\mathbf{x}\| + \epsilon \\ &\leq (\mathbf{w}^\top \mathbf{x} + b) + \epsilon + \epsilon \\ &\leq 12\sqrt{\delta \log(t)/d}. \end{aligned}$$

Hence, any such point lies within distance $12\sqrt{\delta \log(t)/d}$.

Step 5: Using the core inequality. We showed in the steps above that with probability at least $1/5$, none of the events $\mathcal{E}_1, \mathcal{E}_1, \mathcal{E}_1, \mathcal{E}_4$ happen. We will work in such an event, *i.e.*, the event when none of $\mathcal{E}_1, \mathcal{E}_1, \mathcal{E}_1, \mathcal{E}_4$ happen.

Let h denote the minimum number of hyperplanes needed to separate the dataset that we have constructed. We set $t = 8h$. Then, we showed in Step 3 that any hyperplane at distance more than $4\sqrt{\delta \log(t)/(d-1)}$ from a cluster center can only separate $n/t = n/8h$ pairs of points, whereas there are a total of $n/4$ pairs of opposite label points in each cluster. Hence, we need at least one hyperplane to pass within distance $4\sqrt{\delta \log(t)/(d-1)}$ of the cluster center, otherwise there would be at least $n/8$ pairs of opposite label points that would not be separated. On the other hand, we showed in Step 4 that a hyperplane can pass within distance $4\sqrt{\delta \log(t)/(d-1)}$ of only $2C_5\sqrt{\delta \log t}$ fraction of the cluster centers. Now, we are ready to use the core inequality (H.7) to get that

$$h \geq \frac{\sqrt{n} \times 1}{2C_5\sqrt{\delta \log t} \sqrt{n}}.$$

This gives that $h = \Omega(1/(\sqrt{\delta \log(1/\delta)}))$.

For the case when we need to separate every pair of points (Theorem 12.16), the only difference in the inequality above is that we would need at least $\log_2(n - \frac{n}{8})$ hyperplanes per

cluster (instead of 1) that need to pass within distance $4\sqrt{\frac{\delta \log t}{d-1}}$ of the cluster center. This is because separating m points needs $\log_2 m$ hyperplanes. Substituting this in the inequality (H.7) gives the bound in Theorem 12.16. $\square$

H.2.1 Proof of Lemma H.6

Proof. Let $\mathcal{P}_\delta$ be a δ packing of the unit sphere $\mathcal{S}^{d-1}$. Let $\mathcal{B}^d$ denote the unit ball, *i.e.*, the sphere $\mathcal{S}^{d-1}$ along with its interior. Then, if we draw balls of radius δ around each point in the packing, they will not intersect and all of them will be completely contained inside the ball of radius $1 + \delta$. Thus, we get the following

$$\begin{aligned} |\mathcal{P}_\delta| \times \text{Vol}(\delta \mathcal{B}^d) &\leq \text{Vol}((1 + \delta) \mathcal{B}^d) \\ \implies |\mathcal{P}_\delta| &\leq \frac{\text{Vol}((1 + \delta) \mathcal{B}^d)}{\text{Vol}(\delta \mathcal{B}^d)} \\ &= \left(\frac{1 + \delta}{\delta}\right)^d \\ &\leq \left(\frac{2}{\delta}\right)^d. \end{aligned}$$

Next, let $\mathcal{C}_\delta$ be a δ -covering of the sphere $\mathcal{S}^{d-1}$. Any point on the sphere $(1 - \delta) \mathcal{S}^{d-1}$ is at distance δ of some point on $\mathcal{S}^{d-1}$. That point is, in turn, at distance within δ of some point in $\mathcal{C}_\delta$. Hence, by the triangle inequality, every point on the sphere $(1 - \delta) \mathcal{S}^{d-1}$ is within distance 2δ of some point in $\mathcal{C}_\delta$. Similarly, it can be shown that every point on the sphere $(1 + \delta) \mathcal{S}^{d-1}$ is within distance 2δ of some point in $\mathcal{C}_\delta$. Draw balls of radius 2δ around each point in the covering $\mathcal{C}_\delta$. Then, these balls cover all the points within distance δ of the sphere $\mathcal{S}^{d-1}$. Thus, we get the following

$$\begin{aligned} |\mathcal{C}_\delta| \times \text{Vol}(2\delta \mathcal{B}^d) &\geq \text{Vol}(((1 + \delta) \mathcal{B}^d) \setminus ((1 - \delta) \mathcal{B}^d)), \\ \text{which implies } |\mathcal{C}_\delta| &\geq \frac{\text{Vol}((1 + \delta) \mathcal{B}^d) - \text{Vol}((1 - \delta) \mathcal{B}^d)}{\text{Vol}(2\delta \mathcal{B}^d)} \\ &= \frac{(1 + \delta)^d - (1 - \delta)^d}{(2\delta)^d} \\ &\geq \frac{(1 + \delta)^d - 1}{(2\delta)^d} \\ &\geq \frac{1 + d\delta - 1}{(2\delta)^d} \\ &= \frac{d\delta}{(2\delta)^d} \end{aligned}$$

$$\geq \frac{1}{(4\delta)^{(d-1)}}.$$

Combining with the following inequality (Vershynin, 2018a, Lemma 4.2.8):

$$\mathcal{C}_\delta \leq \mathcal{P}_\delta,$$

we get

$$\left(\frac{1}{4\delta}\right)^{d-1} \leq \mathcal{C}_\delta \leq \mathcal{P}_\delta \leq \left(\frac{2}{\delta}\right)^d.$$

□

H.3 Proofs from Section 12.6

H.3.1 Proof of Theorem 12.12

$\mathcal{H} \downarrow, \mathcal{P} \rightarrow$	p_1	p_2	p_3	$\dots$	$p_{ \mathcal{P} }$
h_1	1	0	0		0
h_2	1	1	1		1
h_3	0	1	0		1
$\vdots$					
$h_{ \mathcal{H} }$	0	1	0		1

Figure H.2: (Lower bound) Here, if there are r rows and c columns, and $2^r < c$, then there will be two columns with the same pattern of 0's and 1's. Noting that $r = |\mathcal{H}|$ and $c = |\mathcal{P}|$ gives us the lower bound.

Proof. Let $\mathcal{A}$ output models from the hypothesis set $\mathcal{H}$. Then, we will essentially try to lower bound $\log_2 |\mathcal{H}|$, since we will need at least $\log_2 |\mathcal{H}|$ bits to represent the models.

The first lower bound is easy to see: For a fixed set of n points, there can be 2^n possible assignments of labels. Hence we need at least 2^n hypothesis in our hypothesis class. Hence we need at least n bits to memorize our dataset.

A lower bound with dependence on δ is also not difficult to get. Here is how we create our dataset: Create a δ -packing $\mathcal{P}_\delta$ of the set $\mathcal{S}$. Choose any n points from this packing. In fact, we will only concentrate on the first and the second point in our dataset. Assume that we have a hypothesis set $\mathcal{H}$ such that this can memorize these two points. This means that no matter which two points we choose from the packing and what labels we assign them,

there will be a hypothesis in $\mathcal{H}$ that gives the chosen points the assigned labels. Now, if $2^{|\mathcal{H}|} < |\mathcal{P}_\delta|$, then there exist at least one pair of points in the packing $\mathcal{P}_\delta$ such that every hypothesis assigns the two points the same labels. To see how, consider Figure H.2. There can be at most $2^{|\mathcal{H}|}$ unique columns in that table. Hence, if $2^{|\mathcal{H}|} < |\mathcal{P}_\delta|$, then there would exist two points which have the same columns. We choose those two points and give them opposite labels, then this would contradict memorization, since all the hypotheses in $\mathcal{H}$ give them both the same label. This proves that $2^{|\mathcal{H}|} \geq |\mathcal{P}_\delta|$, or $\log_2 |\mathcal{H}| \geq \log_2 \log_2 |\mathcal{P}_\delta|$. Combined with the first lower bound, we get that we need at least $\max(n, \log_2 \log_2 |\mathcal{P}_\delta|)$ bits. $\square$

H.3.2 Proof of Theorem 12.13

$\mathcal{H} \downarrow, \mathcal{C} \rightarrow$	c_1	c_2	c_3	$\dots$	$c_{ \mathcal{C} }$
h_1	1	0	0		0
h_2	1	1	1		1
h_3	0	1	0		1
$\vdots$					
$h_{ \mathcal{H} }$	0	1	0		1

Figure H.3: (Upper bound) Each entry of this table is i.i.d. Bernoulli with probability 0.5. All we need to do is to ensure that if we choose any n columns above and any labeling in $\{0, 1\}^n$, there exists one row that has that same labeling.

Proof. Create a $\delta/2$ covering $\mathcal{C}_{\delta/2}$ of the set $\mathcal{S}$. We can partition the sphere into $|\mathcal{C}_{\delta/2}|$ regions of diameter less than δ : Any point that lies within distance $\delta/2$ to any point in $\mathcal{C}_{\delta/2}$ is included in the region of that point. If a point lies within distance $\delta/2$ of multiple points in $\mathcal{C}_{\delta/2}$, then assign it to any one of the regions arbitrarily. Since by assumption, no two points in our dataset are within distance δ , the benefit of this partition is that every pair of points in our dataset will lie in different regions of the sphere. Consider the set of all the hypotheses h_i that assign the same label to all the points which lie in a region of the partition. (If $\mathcal{S}$ is a sphere, then these hypotheses look like a football with black and white regions: black region is 0, white is 1). There will be $2^{|\mathcal{C}_{\delta/2}|}$ such hypotheses. We will only select a random subset of this large set and denote it by $\mathcal{H}$. We will show that $\log_2 |\mathcal{H}| = \mathcal{O}(n + \log_2 \log_2 |\mathcal{C}_{\delta/2}|)$ will suffice for memorization.

For any particular choice of n points and n labels, the probability that none of the $|\mathcal{H}|$ hypothesis have that choice of labels is $(1 - 2^{-n})^{|\mathcal{H}|}$. Taking a union bound over all the

$\binom{|\mathcal{C}_{\delta/2}|}{n}$ choices of points and 2^n choices of labels, we need

$$\binom{|\mathcal{C}_{\delta/2}|}{n} 2^n (1 - 2^{-n})^{|\mathcal{H}|} < 1,$$

to ensure that there exists one $\mathcal{H}$ for which we can memorize any n points. Noting that $1 - 2^{-n} < e^{-2^{-n}}$ and $\binom{|\mathcal{C}_{\delta/2}|}{n} < |\mathcal{C}_{\delta/2}|^n$, we see that it is sufficient to have

$$(2|\mathcal{C}_{\delta/2}|)^n e^{-2^{-n}|\mathcal{H}|} < 1.$$

Solving this, we get that $\log_2 |\mathcal{H}| > n + \log_2 n + \log_2 \log_e 2|\mathcal{C}_{\delta/2}|$ is sufficient, *i.e.*, we need $O(n + \log_2 \log_2 |\mathcal{C}_{\delta/2}|)$ bits to memorize n points. $\square$

REFERENCES

-
- Ahn, Kwangjun, Chulhee Yun, and Suvrit Sra. 2020. Sgd with shuffling: optimal rates without component convexity and large epoch requirements. *Advances in Neural Information Processing Systems* 33:17526–17535.
- Akyürek, Ekin, Dale Schuurmans, Jacob Andreas, Tengyu Ma, and Denny Zhou. 2022. What learning algorithm is in-context learning? investigations with linear models. *arXiv preprint arXiv:2211.15661*.
- Andoni, Alexandr, Rina Panigrahy, Gregory Valiant, and Li Zhang. 2014. Learning polynomials with neural networks. In *Proceedings of the 31st international conference on machine learning*, ed. Eric P. Xing and Tony Jebara, vol. 32 of *Proceedings of Machine Learning Research*, 1908–1916. Beijing, China: PMLR.
- Arpit, Devansh, Stanisław Jastrzębski, Nicolas Ballas, David Krueger, Emmanuel Bengio, Maxinder S Kanwal, Tegan Maharaj, Asja Fischer, Aaron Courville, Yoshua Bengio, et al. 2017. A closer look at memorization in deep networks. In *International conference on machine learning*, 233–242. PMLR.
- Arratia, Richard, and Louis Gordon. 1989. Tutorial on large deviations for the binomial distribution. *Bulletin of mathematical biology* 51(1):125–131.
- Ba, Jimmy Lei, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450*.
- Bach, Francis. 2017. Breaking the curse of dimensionality with convex neural networks. *The Journal of Machine Learning Research* 18(1):629–681.
- Balduzzi, David, Marcus Frean, Lennox Leary, JP Lewis, Kurt Wan-Duo Ma, and Brian McWilliams. 2017. The shattered gradients problem: If resnets are the answer, then what is the question? In *International conference on machine learning*, 342–350. PMLR.
- Ball, Keith, et al. 1997. An elementary introduction to modern convex geometry. *Flavors of geometry* 31:1–58.
- Barron, Andrew R. 1993a. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information theory* 39(3):930–945.
- Barron, A.R. 1993b. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information Theory* 39(3):930–945.

- Bartlett, Peter L. 1998. The sample complexity of pattern classification with neural networks: the size of the weights is more important than the size of the network. *IEEE transactions on Information Theory* 44(2):525–536.
- Basak, Anirban, and Mark Rudelson. 2018. Sharp transition of the invertibility of the adjacency matrices of sparse random graphs.
- Baum, Eric B. 1988. On the capabilities of multilayer perceptrons. *Journal of complexity* 4(3):193–215.
- Belkin, Mikhail, Daniel Hsu, and Ji Xu. 2020. Two models of double descent for weak features. *SIAM Journal on Mathematics of Data Science* 2(4):1167–1180.
- Bertsekas, Dimitri P. 2011. Incremental gradient, subgradient, and proximal methods for convex optimization: A survey. *Optimization for Machine Learning* 2010(1-38):3.
- Bjorck, Nils, Carla P Gomes, Bart Selman, and Kilian Q Weinberger. 2018. Understanding batch normalization. *Advances in neural information processing systems* 31.
- Blalock, Davis, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John Gutttag. 2020. What is the state of neural network pruning? *Proceedings of machine learning and systems* 2: 129–146.
- Borsellino, Antonio, and Alberto Gamba. 1961. An outline of a mathematical theory of papa. *Il Nuovo Cimento (1955-1965)* 20(Suppl 2):221–231.
- Bottou, Léon. 2009. Curiously fast convergence of some stochastic gradient descent algorithms. In *Proceedings of the symposium on learning and data science, paris*, vol. 8, 2624–2633. Citeseer.
- Brown, Tom, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33:1877–1901.
- Bubeck, Sébastien, Ronen Eldan, Yin Tat Lee, and Dan Mikulincer. 2020. Network size and weights size for memorization with two-layers neural networks. *arXiv preprint arXiv:2006.02855*.
- Bubeck, Sébastien, et al. 2015. Convex optimization: Algorithms and complexity. *Foundations and Trends® in Machine Learning* 8(3-4):231–357.

- Caron, Richard, and Tim Traynor. 2005. The zero set of a polynomial. *WSMR Report* 05–02.
- Cha, Jaeyoung, Jaewook Lee, and Chulhee Yun. 2023. Tighter lower bounds for shuffling sgd: Random permutations and beyond. *arXiv preprint arXiv:2303.07160*.
- Charles, Zachary, Shashank Rajput, Stephen Wright, and Dimitris Papailiopoulos. 2019. Convergence and margin of adversarial training on separable data. *arXiv preprint arXiv:1905.09209*.
- Charton, François. 2021. Linear algebra with transformers. *arXiv preprint arXiv:2112.01898*.
- Chen, Zejia, Fabing Duan, Francois Chapeau-Blondeau, and Derek Abbott. 2022. Training threshold neural networks by extreme learning machine and adaptive stochastic resonance. *Physics Letters A* 432:128008.
- Cheng, Yu, Duo Wang, Pan Zhou, and Tao Zhang. 2019. A Survey of Model Compression and Acceleration for Deep Neural Networks. *arXiv:1710.09282 [cs]*.
- Chowdhery, Aakanksha, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. 2022. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*.
- Chung, Hyung Won, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Eric Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. 2022. Scaling instruction-finetuned language models. *arXiv preprint arXiv:2210.11416*.
- Clark, Kevin, Minh-Thang Luong, Urvashi Khandelwal, Christopher D Manning, and Quoc V Le. 2019. Bam! born-again multi-task networks for natural language understanding. *arXiv preprint arXiv:1907.04829*.
- Cosentino, Justin, Federico Zaiter, Dan Pei, and Jun Zhu. 2019. The search for sparse, robust neural networks. 1912.02386.
- Courbariaux, Matthieu, Yoshua Bengio, and Jean-Pierre David. 2015. Binaryconnect: Training deep neural networks with binary weights during propagations. *arXiv preprint arXiv:1511.00363*.
- Cover, Thomas M. 1965. Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *IEEE transactions on electronic computers*.

- Dasgupta, Ishita, Andrew K Lampinen, Stephanie CY Chan, Antonia Creswell, Dharshan Kumaran, James L McClelland, and Felix Hill. 2022. Language models show human-like content effects on reasoning. *arXiv preprint arXiv:2207.07051*.
- De Sa, Christopher M. 2020. Random reshuffling is not always better. *Advances in Neural Information Processing Systems* 33.
- Dehghani, Mostafa, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. 2018. Universal transformers. *arXiv preprint arXiv:1807.03819*.
- Deng, Jia, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *2009 ieee conference on computer vision and pattern recognition*, 248–255. Ieee.
- Deng, Lei, Guoqi Li, Song Han, Luping Shi, and Yuan Xie. 2020. Model compression and hardware acceleration for neural networks: A comprehensive survey. *Proceedings of the IEEE* 108(4):485–532.
- Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.
- Dieuleveut, Aymeric, and Kumar Kshitij Patel. 2019. Communication trade-offs for local-sgd with large step size. *Advances in Neural Information Processing Systems* 32:13601–13612.
- Diffenderfer, James, and Bhavya Kailkhura. 2021. Multi-prize lottery ticket hypothesis: Finding accurate binary neural networks by pruning a randomly weighted network. *arXiv preprint arXiv:2103.09377*.
- Dinh, Tuan, Jy-yong Sohn, Shashank Rajput, Timothy Ossowski, Yifei Ming, Junjie Hu, Dimitris Papailiopoulos, and Kangwook Lee. 2022a. Utilizing language-image pretraining for efficient and robust bilingual word alignment. *arXiv preprint arXiv:2205.11616*.
- Dinh, Tuan, Yuchen Zeng, Ruisu Zhang, Ziqian Lin, Michael Gira, Shashank Rajput, Jy-yong Sohn, Dimitris Papailiopoulos, and Kangwook Lee. 2022b. Lift: Language-interfaced fine-tuning for non-language machine learning tasks. *Advances in Neural Information Processing Systems* 35:11763–11784.
- Donahue, Jeff, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. 2014. Decaf: A deep convolutional activation feature for generic visual recognition. In *International conference on machine learning*, 647–655. PMLR.

Dosovitskiy, Alexey, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. 2020. An image is worth 16x16 words: Transformers for image recognition at scale. In *International conference on learning representations*.

———. 2021. An image is worth 16x16 words: Transformers for image recognition at scale. In *International conference on learning representations*.

Du, Simon S., Xiyu Zhai, Barnabas Poczos, and Aarti Singh. 2018. Gradient descent provably optimizes over-parameterized neural networks.

Ergen, Tolga, Halil Ibrahim Gulluk, Jonathan Lacotte, and Mert Pilanci. 2023. Globally optimal training of neural networks with threshold activation functions. In *International conference on learning representations*.

Frankle, Jonathan, and Michael Carbin. 2018a. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*.

———. 2018b. The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks. In *International Conference on Learning Representations*.

Frankle, Jonathan, Gintare Karolina Dziugaite, Daniel Roy, and Michael Carbin. 2020a. Linear mode connectivity and the lottery ticket hypothesis. In *International conference on machine learning*, 3259–3269. PMLR.

Frankle, Jonathan, Gintare Karolina Dziugaite, Daniel M. Roy, and Michael Carbin. 2020b. Linear Mode Connectivity and the Lottery Ticket Hypothesis. *arXiv:1912.05671 [cs, stat]*.

Frankle, Jonathan, David J Schwab, and Ari S Morcos. 2020c. Training batchnorm and only batchnorm: On the expressive power of random features in cnns. *arXiv preprint arXiv:2003.00152*.

Garg, Shivam, Dimitris Tsipras, Percy Liang, and Gregory Valiant. 2022. What can transformers learn in-context? a case study of simple function classes. In *Advances in neural information processing systems*.

Ghorbani, Behrooz, Song Mei, Theodor Misiakiewicz, and Andrea Montanari. 2019. Linearized two-layers neural networks in high dimension.

Giannou, Angeliki, Shashank Rajput, and Dimitris Papailiopoulos. 2023. The expressive power of tuning only the norm layers. *arXiv preprint arXiv:2302.07937*.

Giannou, Angeliki, Shashank Rajput, Jy-yong Sohn, Kangwook Lee, Jason D Lee, and Dimitris Papailiopoulos. Looped transformers as programmable computers. In *Iclr 2023 workshop on mathematical and empirical understanding of foundation models*.

Gitman, Igor, and Boris Ginsburg. 2017. Comparison of batch normalization and weight normalization algorithms for the large-scale image classification. *arXiv preprint arXiv:1709.08145*.

Gribonval, Rémi, Gitta Kutyniok, Morten Nielsen, and Felix Voigtlaender. 2022. Approximation spaces of deep neural networks. *Constructive approximation* 55(1):259–367.

Guo, Demi, Alexander M Rush, and Yoon Kim. 2020. Parameter-efficient transfer learning with diff pruning. *arXiv preprint arXiv:2012.07463*.

Gurbuzbalaban, M, A Ozdaglar, and PA Parrilo. 2019a. Convergence rate of incremental gradient and incremental newton methods. *SIAM Journal on Optimization* 29(4):2542–2565.

Gürbüzbalaban, Mert, Asu Ozdaglar, and PA Parrilo. 2015. Why random reshuffling beats stochastic gradient descent. *Mathematical Programming* 1–36.

Gurbuzbalaban, Mert, Asuman Ozdaglar, Nuri Denizcan Vanli, and Stephen J Wright. 2019b. Randomness and permutations in coordinate descent methods. *Mathematical Programming* 1–28.

Gurobi Optimization, LLC. 2020. Gurobi optimizer reference manual.

Guruswami, Venkatesan, Atri Rudra, and Madhu Sudan. 2012. Essential coding theory. *Draft available at <http://www.cse.buffalo.edu/~atri/courses/coding-theory/book>*.

Haddadpour, Farzin, Mohammad Mahdi Kamani, Mehrdad Mahdavi, and Viveck Cadambe. 2019. Local sgd with periodic averaging: Tighter analysis and adaptive synchronization. *Advances in Neural Information Processing Systems* 32:11082–11094.

Haddadpour, Farzin, and Mehrdad Mahdavi. 2019. On the convergence of local descent methods in federated learning. *arXiv preprint arXiv:1910.14425*.

Han, Song, Huizi Mao, and William J Dally. 2015a. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*.

Han, Song, Huizi Mao, and William J. Dally. 2016. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. *arXiv:1510.00149 [cs]*.

Han, Song, Jeff Pool, John Tran, and William J Dally. 2015b. Learning both weights and connections for efficient neural networks. *arXiv preprint arXiv:1506.02626*.

Hanin, Boris. 2019. Universal function approximation by deep neural nets with bounded width and relu activations. *Mathematics* 7(10):992.

Haochen, Jeff, and Suvrit Sra. 2019a. Random shuffling beats sgd after finite epochs. In *International conference on machine learning*, 2624–2633. PMLR.

———. 2019b. Random shuffling beats SGD after finite epochs. In *International conference on machine learning*, 2624–2633.

Hardt, Moritz, and Tengyu Ma. 2017. Identity matters in deep learning. In *5th international conference on learning representations, ICLR 2017, toulon, france, april 24-26, 2017, conference track proceedings*.

Hardt, Moritz, Ben Recht, and Yoram Singer. 2016. Train faster, generalize better: Stability of stochastic gradient descent. In *International conference on machine learning*, 1225–1234. PMLR.

Hassibi, Babak, and David G. Stork. 1993a. Second order derivatives for network pruning: Optimal Brain Surgeon. In *Advances in Neural Information Processing Systems 5*, ed. S. J. Hanson, J. D. Cowan, and C. L. Giles, 164–171. Morgan-Kaufmann.

———. 1993b. Second order derivatives for network pruning: Optimal Brain Surgeon. In *Advances in Neural Information Processing Systems 5*, ed. S. J. Hanson, J. D. Cowan, and C. L. Giles, 164–171. Morgan-Kaufmann.

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR* abs/1502.01852. 1502.01852.

———. 2016a. Deep residual learning for image recognition. In *Proceedings of the ieee conference on computer vision and pattern recognition*, 770–778.

———. 2016b. Identity mappings in deep residual networks. In *European conference on computer vision*, 630–645. Springer.

He, Yihui, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. 2018. Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the european conference on computer vision (eccv)*, 784–800.

- He, Yihui, Xiangyu Zhang, and Jian Sun. 2017. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE international conference on computer vision*, 1389–1397.
- Hochreiter, Sepp, and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9(8):1735–1780.
- Hoeffler, Torsten, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. 2021. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *The Journal of Machine Learning Research* 22(1):10882–11005.
- Horn, Roger A., and Charles R. Johnson. 1991. *Topics in matrix analysis*. Cambridge University Press.
- Houlsby, Neil, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Larousilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for nlp.
- Hsu, Daniel, Clayton H Sanford, Rocco Servedio, and Emmanouil Vasileios Vlatakis-Gkaragkounis. 2021. On the approximation power of two-layer networks of random relus. In *Conference on learning theory*, 2423–2461. PMLR.
- Huang, Guang-Bin. 2003. Learning capability and storage capacity of two-hidden-layer feedforward networks. *IEEE transactions on neural networks* 14(2):274–281.
- Huang, Shih-Chi, Yih-Fang Huang, et al. 1991. Bounds on the number of hidden neurons in multilayer perceptrons. *IEEE transactions on neural networks* 2(1):47–55.
- Huang, Xun, and Serge Belongie. 2017. Arbitrary style transfer in real-time with adaptive instance normalization. In *Proceedings of the IEEE international conference on computer vision*, 1501–1510.
- Hubara, Itay, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Binarized neural networks. In *Proceedings of the 30th international conference on neural information processing systems*, 4114–4122.
- . 2017. Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research* 18(1):6869–6898.
- Hutchins, DeLesley, Imanol Schlag, Yuhuai Wu, Ethan Dyer, and Behnam Neyshabur. 2022. Block-recurrent transformers. *arXiv preprint arXiv:2203.07852*.

Idelbayev, Yerlan, and Miguel A Carreira-Perpinán. 2020. Low-rank compression of neural nets: Learning the rank of each layer. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 8049–8059.

Ioffe, Sergey, and Christian Szegedy. 2015a. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, 448–456. pmlr.

———. 2015b. Batch normalization: Accelerating deep network training by reducing internal covariate shift.

Jacot, Arthur, Franck Gabriel, and Clément Hongler. 2018. Neural tangent kernel: Convergence and generalization in neural networks. *Advances in neural information processing systems* 31.

Jain, Prateek, Dheeraj Nagaraj, and Praneeth Netrapalli. 2019. Making the last iterate of sgd information theoretically optimal. In *Conference on learning theory*, 1752–1755. PMLR.

Ji, Ziwei, Matus Telgarsky, and Ruicheng Xian. 2019. Neural tangent kernels, transportation mappings, and universal approximation.

Jiang, Tao, N.D. Sidiropoulos, and J.M.F. ten Berge. 2001. Almost-sure identifiability of multidimensional harmonic retrieval. *IEEE Transactions on Signal Processing* 49(9):1849–1859.

Jiang, Yiding, Vaishnavh Nagarajan, Christina Baek, and J Zico Kolter. 2022. Assessing generalization of sgd via disagreement. In *International conference on learning representations*.

Kairouz, Peter, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, Rafael G. L. D’Oliveira, Hubert Eichner, Salim El Rouayheb, David Evans, Josh Gardner, Zachary Garrett, Adrià Gascón, Badih Ghazi, Phillip B. Gibbons, Marco Gruteser, Zaid Harchaoui, Chaoyang He, Lie He, Zhouyuan Huo, Ben Hutchinson, Justin Hsu, Martin Jaggi, Tara Javidi, Gauri Joshi, Mikhail Khodak, Jakub Konečný, Aleksandra Korolova, Farinaz Koushanfar, Sanmi Koyejo, Tancrède Lepoint, Yang Liu, Prateek Mittal, Mehryar Mohri, Richard Nock, Ayfer Özgür, Rasmus Pagh, Hang Qi, Daniel Ramage, Ramesh Raskar, Mariana Raykova, Dawn Song, Weikang Song, Sebastian U. Stich, Ziteng Sun, Ananda Theertha Suresh, Florian Tramèr, Praneeth Vepakomma, Jianyu Wang, Li Xiong, Zheng Xu, Qiang Yang, Felix X. Yu, Han Yu, and Sen Zhao. 2021. *Advances*

and open problems in federated learning. *Foundations and Trends® in Machine Learning* 14(1–2):1–210.

Kamalakara, Siddhartha Rao, Acyr Locatelli, Bharat Venkitesh, Jimmy Ba, Yarin Gal, and Aidan N. Gomez. 2022. Exploring low rank training of deep neural networks.

Karimi, Hamed, Julie Nutini, and Mark Schmidt. 2016. Linear convergence of gradient and proximal-gradient methods under the Polyak-Łojasiewicz condition. In *Joint european conference on machine learning and knowledge discovery in databases*, 795–811. Springer.

Karimireddy, Sai Praneeth, Satyen Kale, Mehryar Mohri, Sashank Reddi, Sebastian Stich, and Ananda Theertha Suresh. 2020. Scaffold: Stochastic controlled averaging for federated learning. In *International conference on machine learning*, 5132–5143. PMLR.

Karmarkar, Narendra, Richard M. Karp, George S. Lueker, and Andrew M. Odlyzko. 1986. Probabilistic Analysis of Optimum Partitioning. *Journal of Applied Probability* 23(3): 626–645.

Karp, Richard M. 1972. Reducibility among Combinatorial Problems. In *Complexity of Computer Computations: Proceedings of a symposium on the Complexity of Computer Computations, held March 20–22, 1972.*, ed. Raymond E. Miller, James W. Thatcher, and Jean D. Bohlinger, 85–103. The IBM Research Symposia Series, Boston, MA: Springer US.

Kenton, Jacob Devlin Ming-Wei Chang, and Lee Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of naacl-hlt*, 4171–4186.

Khaled, Ahmed, Konstantin Mishchenko, and Peter Richtárik. 2020. Tighter theory for local sgd on identical and heterogeneous data. In *International conference on artificial intelligence and statistics*, 4519–4529. PMLR.

Khan, Salman, Muzammal Naseer, Munawar Hayat, Syed Waqas Zamir, Fahad Shahbaz Khan, and Mubarak Shah. 2022. Transformers in vision: A survey. *ACM computing surveys (CSUR)* 54(10s):1–41.

Kidger, Patrick, and Terry Lyons. 2020. Universal approximation with deep narrow networks. In *Conference on learning theory*, 2306–2327. PMLR.

Klusowski, Jason M., and Andrew R. Barron. 2016. Approximation by combinations of relu and squared relu ridge functions with ℓ^1 and ℓ^0 controls.

- Klusowski, Jason M, and Andrew R Barron. 2018. Approximation by combinations of relu and squared relu ridge functions with ℓ^1 and ℓ^0 controls. *IEEE Transactions on Information Theory* 64(12):7649–7656.
- Kohler, Jonas, Hadi Daneshmand, Aurelien Lucchi, Thomas Hofmann, Ming Zhou, and Klaus Neymeyr. 2019. Exponential convergence rates for batch normalization: The power of length-direction decoupling in non-convex optimization. In *The 22nd international conference on artificial intelligence and statistics*, 806–815. PMLR.
- Koloskova, Anastasia, Nicolas Loizou, Sadra Boreiri, Martin Jaggi, and Sebastian Stich. 2020. A unified theory of decentralized sgd with changing topology and local updates. In *International conference on machine learning*, 5381–5393. PMLR.
- Konečný, Jakub, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. 2016. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*.
- Kotsovsky, Vladyslav, Fedir Geche, and Anatoliy Batyuk. 2020. Bithreshold neural network classifier. In *2020 IEEE 15th international conference on computer sciences and information technologies (csit)*, vol. 1, 32–35. IEEE.
- Kowalczyk, Adam. 1997. Estimates of storage capacity of multilayer perceptron with threshold logic hidden units. *Neural networks* 10(8):1417–1433.
- Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton. 2017. Imagenet classification with deep convolutional neural networks. *Communications of the ACM* 60(6):84–90.
- Lai, Zehua, and Lek-Heng Lim. 2020. Recht-Ré noncommutative arithmetic-geometric mean conjecture is false. In *International conference on machine learning*.
- Lecun, Y., L. Bottou, Y. Bengio, and P. Haffner. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* 86(11):2278–2324.
- LeCun, Yann, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* 86(11):2278–2324.
- LeCun, Yann, John S. Denker, and Sara A. Solla. 1990. Optimal Brain Damage. In *Advances in Neural Information Processing Systems 2*, ed. D. S. Touretzky, 598–605. Morgan-Kaufmann.
- LeCun, Yann A, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. 2012. Efficient backprop. In *Neural networks: Tricks of the trade*, 9–48. Springer.

- Lee, Ching-Pei, and Stephen J Wright. 2019. Random permutations fix a worst case for cyclic coordinate descent. *IMA Journal of Numerical Analysis* 39(3):1246–1275.
- Lee, Kangwook, Maximilian Lam, Ramtin Pedarsani, Dimitris Papailiopoulos, and Kannan Ramchandran. 2017. Speeding up distributed machine learning using codes. *IEEE Transactions on Information Theory* 64(3):1514–1529.
- Levin, Asriel U., Todd K. Leen, and John E. Moody. 1994. Fast Pruning Using Principal Components. In *Advances in Neural Information Processing Systems 6*, ed. J. D. Cowan, G. Tesauro, and J. Alspector, 35–42. Morgan-Kaufmann.
- Lewkowycz, Aitor, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, et al. 2022. Solving quantitative reasoning problems with language models. *arXiv preprint arXiv:2206.14858*.
- Li, Hao, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. 2016. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*.
- Li, Tian, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. 2020a. Federated learning: Challenges, methods, and future directions. *IEEE Signal Processing Magazine* 37(3):50–60.
- Li, Tian, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. 2020b. Federated optimization in heterogeneous networks. In *Proceedings of machine learning and systems*, ed. I. Dhillon, D. Papailiopoulos, and V. Sze, vol. 2, 429–450.
- Li, Xiang, Kaixuan Huang, Wenhao Yang, Shusen Wang, and Zhihua Zhang. 2020c. On the convergence of fedavg on non-iid data. In *International conference on learning representations*.
- Lifchitz, Yann, Yannis Avrithis, Sylvaine Picard, and Andrei Bursuc. 2019. Dense classification and implanting for few-shot learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 9258–9267.
- Lindner, David, János Kramár, Matthew Rahtz, Thomas McGrath, and Vladimir Mikulik. 2023. Tracr: Compiled transformers as a laboratory for interpretability. *arXiv preprint arXiv:2301.05062*.
- Liu, Bingbin, Jordan T Ash, Surbhi Goel, Akshay Krishnamurthy, and Cyril Zhang. 2022. Transformers learn shortcuts to automata. *arXiv preprint arXiv:2210.10749*.

- Liu, Shengchao, Dimitris Papailiopoulos, and Dimitris Achlioptas. 2020. Bad global minima exist and sgd can reach them. *Advances in Neural Information Processing Systems* 33.
- Liu, Shuangzhe, and OTZ Trenkler. 2008. Hadamard, khatri-rao, kronecker and other matrix products. *International Journal of Information & Systems Sciences* 4.
- Loshchilov, Ilya, and Frank Hutter. 2016. SGDR: stochastic gradient descent with restarts. *CoRR* abs/1608.03983. 1608.03983.
- Lu, Yucheng, Wentao Guo, and Christopher M De Sa. 2022. Grab: Finding provably better data permutations than random reshuffling. *Advances in Neural Information Processing Systems* 35:8969–8981.
- Lueker, George S. 1982. On the Average Difference between the Solutions to Linear and Integer Knapsack Problems. In *Applied Probability-Computer Science: The Interface Volume 1*, ed. Ralph L. Disney and Teunis J. Ott, 489–504. Progress in Computer Science, Boston, MA: Birkhäuser.
- Lueker, George S. 1998a. Exponentially small bounds on the expected optimum of the partition and subset sum problems. *Random Structures & Algorithms* 12(1):51–62.
- Lueker, George S. 1998b. Exponentially small bounds on the expected optimum of the partition and subset sum problems. *Random Structures & Algorithms* 12(1):51–62.
- Luo, Ping, Xinjiang Wang, Wenqi Shao, and Zhanglin Peng. 2018. Towards understanding regularization in batch normalization. *arXiv preprint arXiv:1809.00846*.
- Malach, Eran, Gilad Yehudai, Shai Shalev-Schwartz, and Ohad Shamir. 2020a. Proving the lottery ticket hypothesis: Pruning is all you need. In *International conference on machine learning*, 6682–6691. PMLR.
- . 2020b. Proving the lottery ticket hypothesis: Pruning is all you need. In *International conference on machine learning*, 6682–6691. PMLR.
- Mavaddat, Farhad, and Behrooz Parhami. 1988. Urisc: the ultimate reduced instruction set computer. *International Journal of Electrical Engineering Education* 25(4):327–334.
- McCulloh, WS, and W Pitts. 1943. A logical calculus of the ideas immanent in neural nets. *Bull Math. Biophys* 5:133–137.

- McMahan, Brendan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agueray Arcas. 2017. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, 1273–1282. PMLR.
- Meng, Qi, Wei Chen, Yue Wang, Zhi-Ming Ma, and Tie-Yan Liu. 2017. Convergence analysis of distributed stochastic gradient descent with shuffling. *arXiv preprint arXiv:1709.10432*.
- . 2019. Convergence analysis of distributed stochastic gradient descent with shuffling. *Neurocomputing* 337:46–57.
- Merrill, William, Ashish Sabharwal, and Noah A Smith. 2022. Saturated transformers are constant-depth threshold circuits. *Transactions of the Association for Computational Linguistics* 10:843–856.
- Mishchenko, Konstantin, Ahmed Khaled, and Peter Richtárik. 2020. Random reshuffling: Simple analysis with vast improvements. *arXiv preprint arXiv:2006.05988*.
- . 2021. Proximal and federated random reshuffling. *arXiv preprint arXiv:2102.06704*.
- Mitchison, GJ, and RM Durbin. 1989. Bounds on the learning capacity of some multi-layer networks. *Biological Cybernetics*.
- Morcos, Ari S, David GT Barrett, Neil C Rabinowitz, and Matthew Botvinick. 2018. On the importance of single directions for generalization. *arXiv preprint arXiv:1803.06959*.
- Mortici, Cristinel. 2011. On gospers formula for the gamma function. *Journal of Mathematical Inequalities* 5.
- Mozer, Michael C, and Paul Smolensky. 1989a. Skeletonization: A Technique for Trimming the Fat from a Network via Relevance Assessment. In *Advances in Neural Information Processing Systems 1*, ed. D. S. Touretzky, 107–115. Morgan-Kaufmann.
- . 1989b. Skeletonization: A technique for trimming the fat from a network via relevance assessment. In *Advances in neural information processing systems*, 107–115.
- Mudrakarta, Pramod Kaushik, Mark Sandler, Andrey Zhmoginov, and Andrew Howard. 2018. K for the price of 1: Parameter-efficient multi-task and transfer learning. *arXiv preprint arXiv:1810.10703*.
- Nagaraj, Dheeraj, Prateek Jain, and Praneeth Netrapalli. 2019. Sgd without replacement: Sharper rates for general smooth convex functions. In *International conference on machine learning*, 4703–4711. PMLR.

- Nambiar, Ananthan, Maeve Heflin, Simon Liu, Sergei Maslov, Mark Hopkins, and Anna Ritz. 2020. Transforming the language of life: transformer neural networks for protein prediction tasks. In *Proceedings of the 11th acm international conference on bioinformatics, computational biology and health informatics*, 1–8.
- Narang, Sharan, Eric Undersander, and Gregory Diamos. 2017. Block-sparse recurrent neural networks. *arXiv preprint arXiv:1711.02782*.
- Nedić, Angelia, and Dimitri Bertsekas. 2001. Convergence rate of incremental subgradient algorithms. In *Stochastic optimization: algorithms and applications*, 223–264. Springer.
- Neimark, Daniel, Omri Bar, Maya Zohar, and Dotan Asselmann. 2021. Video transformer network. In *Proceedings of the ieee/cvf international conference on computer vision*, 3163–3172.
- Nesterov, Yurii. 2013. *Introductory lectures on convex optimization: A basic course*, vol. 87. Springer Science & Business Media.
- Neyshabur, Behnam, Zhiyuan Li, Srinadh Bhojanapalli, Yann LeCun, and Nathan Srebro. 2018. The role of over-parametrization in generalization of neural networks. In *International conference on learning representations*.
- . 2019. Towards understanding the role of over-parametrization in generalization of neural networks. In *International conference on learning representations (iclr)*.
- Neyshabur, Behnam, Ryota Tomioka, and Nathan Srebro. 2014. In search of the real inductive bias: On the role of implicit regularization in deep learning. *arXiv preprint arXiv:1412.6614*.
- Ng, Andrew. 2023.
- Nguyen, Lam M., Quoc Tran-Dinh, Dzung T. Phan, Phuong Ha Nguyen, and Marten van Dijk. 2020. A unified convergence analysis for shuffling-type gradient methods. *arXiv preprint arXiv:2002.08246*.
- Ni, Jianmo, Jiacheng Li, and Julian McAuley. 2019. Justifying recommendations using distantly-labeled reviews and fine-grained aspects. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (emnlp-ijcnlp)*, 188–197.
- Nocedal, Jorge, and Stephen Wright. 2006. *Numerical optimization*. Springer Science & Business Media.

Nürnberg, Peter J, Uffe K Wiil, and David L Hicks. 2004. A grand unified theory for structural computing. In *Metainformatics: International symposium, mis 2003, graz, austria, september 17-20, 2003. revised papers*, 1–16. Springer.

Nye, Maxwell, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, et al. 2021. Show your work: Scratchpads for intermediate computation with language models. In *Deep learning for code workshop*.

Orseau, Laurent, Marcus Hutter, and Omar Rivasplata. 2020. Logarithmic pruning is all you need. *Advances in Neural Information Processing Systems* 33.

von Oswald, Johannes, Eyvind Niklasson, Ettore Randazzo, João Sacramento, Alexander Mordvintsev, Andrey Zhmoginov, and Max Vladymyrov. 2022. Transformers learn in-context by gradient descent. *arXiv preprint arXiv:2212.07677*.

Park, Sejun, Jaeho Lee, Chulhee Yun, and Jinwoo Shin. 2020. Provable memorization via deep neural networks using sub-linear parameters. *arXiv preprint arXiv:2010.13363*.

Pensia, Ankit, Shashank Rajput, Alliot Nagle, Harit Vishwakarma, and Dimitris Papailiopoulos. 2020. Optimal lottery tickets via subset sum: Logarithmic over-parameterization is sufficient. *Advances in neural information processing systems* 33:2599–2610.

Perekrestenko, Dmytro, Philipp Grohs, Dennis Elbrächter, and Helmut Bölcskei. 2018. The universal approximation power of finite-width deep relu networks. *arXiv preprint arXiv:1806.01528*.

Pérez, Jorge, Pablo Barceló, and Javier Marinkovic. 2021. Attention is turing-complete. *Journal of Machine Learning Research* 22(75):1–35.

Pinelis, Iosif. 1992. An approach to inequalities for the distributions of infinite-dimensional martingales. In *Probability in banach spaces, 8: Proceedings of the eighth international conference*, 128–134. Springer.

———. 1994. Optimum bounds for the distributions of martingales in banach spaces. *The Annals of Probability* 1679–1706.

Pool, Jeff, Abhishek Sawarkar, and Jay Rodge. 2021. Accelerating inference with sparsity using the nvidia ampere architecture and nvidia tensorrt. <https://developer.nvidia.com/blog/accelerating-inference-with-sparsity-using-ampere-and-tensorrt/>.

Pérez, Jorge, Javier Marinković, and Pablo Barceló. 2019. On the turing completeness of modern neural network architectures.

Qu, Zhaonan, Kaixiang Lin, Jayant Kalagnanam, Zhaojian Li, Jiayu Zhou, and Zhengyuan Zhou. 2020. Federated learning's blessing: Fedavg has linear speedup. *arXiv preprint arXiv:2007.05690*.

Rahimi, Ali, and Benjamin Recht. 2007. Random features for large-scale kernel machines. In *Advances in neural information processing systems*, ed. J. Platt, D. Koller, Y. Singer, and S. Roweis, vol. 20. Curran Associates, Inc.

———. 2008a. Uniform approximation of functions with random bases. *2008 46th Annual Allerton Conference on Communication, Control, and Computing* 555–561.

———. 2008b. Weighted sums of random kitchen sinks: Replacing minimization with randomization in learning. In *Advances in neural information processing systems*, ed. D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, vol. 21. Curran Associates, Inc.

Rajput, S, Z Feng, Z Charles, PL Loh, and D Papailiopoulos. 2019a. Does data augmentation lead to positive margin? In *Proceedings of the 36th international conference on machine learning*.

Rajput, Shashank, Anant Gupta, and Dimitris Papailiopoulos. 2020. Closing the convergence gap of sgd without replacement. In *International conference on machine learning*, 7964–7973. PMLR.

Rajput, Shashank, Kangwook Lee, and Dimitris Papailiopoulos. 2022. Permutation-based sgd: Is random optimal? In *International conference on learning representations*.

Rajput, Shashank, Nikhil Mehta, Anima Singh, Raghunandan H Keshavan, Trung Vu, Lukasz Heldt, Lichan Hong, Yi Tay, Vinh Q Tran, Jonah Samost, et al. 2023. Recommender systems with generative retrieval. *arXiv preprint arXiv:2305.05065*.

Rajput, Shashank, Kartik Sreenivasan, Dimitris Papailiopoulos, and Amin Karbasi. 2021. An exponential improvement on the memorization capacity of deep threshold networks. *Advances in Neural Information Processing Systems* 34:12674–12685.

Rajput, Shashank, Hongyi Wang, Zachary Charles, and Dimitris Papailiopoulos. 2019b. Detox: A redundancy-based framework for faster and more robust gradient aggregation. *Advances in Neural Information Processing Systems* 32.

Rakhlin, Alexander, Ohad Shamir, and Karthik Sridharan. 2012. Making gradient descent optimal for strongly convex stochastic optimization. In *Proceedings of the 29th international conference on machine learning*, 1571–1578.

Ramanujan, Vivek, Mitchell Wortsman, Aniruddha Kembhavi, Ali Farhadi, and Mohammad Rastegari. 2020. What’s Hidden in a Randomly Weighted Neural Network? *arXiv:1911.13299 [cs]*.

Ramesh, Aditya, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. 2021. Zero-shot text-to-image generation. In *International conference on machine learning*, 8821–8831. PMLR.

Rastegari, Mohammad, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. 2016. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European conference on computer vision*, 525–542. Springer.

Recht, Benjamin, and Christopher Ré. 2012a. Beneath the valley of the noncommutative arithmetic-geometric mean inequality: conjectures, case-studies, and consequences. *arXiv preprint arXiv:1202.4184*.

———. 2012b. Toward a noncommutative arithmetic-geometric mean inequality: conjectures, case-studies, and consequences. In *Conference on learning theory*, 11–1.

Recht, Benjamin, and Christopher Re. 2013. Parallel stochastic gradient algorithms for large-scale matrix completion. *Mathematical Programming Computation* 5(2):201–226.

Robbins, Herbert, and Sutton Monro. 1951. A stochastic approximation method. *The annals of mathematical statistics* 400–407.

Rosenblatt, Frank. 1957. *The perceptron, a perceiving and recognizing automaton project para.* Cornell Aeronautical Laboratory.

Sabatelli, Matthia, Mike Kestemont, and Pierre Geurts. 2020. On the transferability of winning tickets in non-natural image datasets. *arXiv preprint arXiv:2005.05232*.

Safran, Itay, and Ohad Shamir. 2020. How good is SGD with random shuffling? In *Conference on learning theory*, 3250–3284. PMLR.

———. 2021. Random shuffling beats SGD only after many epochs on ill-conditioned problems. *arXiv preprint arXiv:2106.06880*.

- Salimans, Tim, and Durk P Kingma. 2016. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *Advances in neural information processing systems* 29.
- Santurkar, Shibani, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. 2018. How does batch normalization help optimization? *Advances in neural information processing systems* 31.
- Sartori, Michael A, and Panos J Antsaklis. 1991. A simple method to derive bounds on the size and to train multilayer neural networks. *IEEE transactions on neural networks* 2(4): 467–471.
- Scarselli, Franco, and Ah Chung Tsoi. 1998. Universal approximation using feedforward neural networks: A survey of some existing methods, and some new results. *Neural networks* 11(1):15–37.
- Schneider, Markus. 2016. Probability inequalities for kernel embeddings in sampling without replacement. In *Artificial intelligence and statistics*, 66–74.
- Serfling, Robert J. 1974. Probability inequalities for the sum in sampling without replacement. *The Annals of Statistics* 39–48.
- Shamir, Ohad. 2016a. Without-replacement sampling for stochastic gradient methods. In *Advances in neural information processing systems* 29, ed. D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, 46–54. Curran Associates, Inc.
- . 2016b. Without-replacement sampling for stochastic gradient methods. In *Advances in neural information processing systems*, 46–54.
- Shen, Zhiqiang, Zechun Liu, and Eric Xing. 2022. Sliced recursive transformer. In *European conference on computer vision*, 727–744. Springer.
- Simons, Taylor, and Dah-Jye Lee. 2019. A review of binarized neural networks. *Electronics* 8(6):661.
- Soelen, R. V., and J. W. Sheppard. 2019. Using winning lottery tickets in transfer learning for convolutional neural networks. In *2019 international joint conference on neural networks (ijcnn)*, 1–8.
- Sontag, Eduardo D. 1990. Remarks on interpolation and recognition using neural nets. In *Nips*, 939–945.

de Souza Pereira Moreira, Gabriel, Sara Rabhi, Jeong Min Lee, Ronay Ak, and Even Oldridge. 2021. Transformers4rec: Bridging the gap between nlp and sequential/session-based recommendation. In *Proceedings of the 15th acm conference on recommender systems*, 143–153.

Spiridonoff, Artin, Alex Olshevsky, and Ioannis Ch Paschalidis. 2020. Local sgd with a communication overhead depending only on the number of workers. *arXiv preprint arXiv:2006.02582*.

Sreenivasan, Kartik, Shashank Rajput, Jy-Yong Sohn, and Dimitris Papailiopoulos. 2022. Finding nearly everything within random binary networks. In *International conference on artificial intelligence and statistics*, 3531–3541. PMLR.

Srivastava, Nitish, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research* 15(1):1929–1958.

Stephen, Merity, Xiong Caiming, Bradbury James, and Richard Socher. 2017. Pointer sentinel mixture models. *Proceedings of ICLR*.

Stich, Sebastian U. 2019. Local sgd converges fast and communicates little. In *International conference on learning representations*.

Stich, Sebastian U, and Sai Praneeth Karimireddy. 2020. The error-feedback framework: Better rates for sgd with delayed gradients and compressed updates. *Journal of Machine Learning Research* 21:1–36.

Stinchcombe, M. 1989. Universal approximation using feed-forward networks with non-sigmoid hidden layer activation functions. *Proc. IJCNN, Washington, DC, 1989* 161–166.

Sun, Ruoyu, Zhi-Quan Luo, and Yinyu Ye. 2019. On the efficiency of random permutation for admm and coordinate descent. *Mathematics of Operations Research*.

Sun, Ruoyu, and Yinyu Ye. 2019. Worst-case complexity of cyclic coordinate descent: $\mathcal{O}(n^2)$ gap with randomized version. *Mathematical Programming* 1–34.

Sun, Yitong, Anna Gilbert, and Ambuj Tewari. 2018. On the approximation properties of random relu features.

Szlendak, Rafał, Alexander Tyurin, and Peter Richtárik. 2021. Permutation compressors for provably faster distributed nonconvex optimization. *arXiv preprint arXiv:2110.03300*.

Tran, Trang H, Lam M Nguyen, and Quoc Tran-Dinh. 2021. Smg: A shuffling gradient-based method with momentum. In *International conference on machine learning*, 10379–10389. PMLR.

Vadera, Sunil, and Salem Ameen. 2022. Methods for pruning deep neural networks. *IEEE Access* 10:63280–63300.

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30.

Vershynin, Roman. 2018a. *High-dimensional probability: An introduction with applications in data science*, vol. 47. Cambridge university press.

———. 2018b. *High-dimensional probability: An introduction with applications in data science*. Cambridge Series in Statistical and Probabilistic Mathematics, Cambridge University Press.

———. 2020. Memory capacity of neural networks with threshold and rectified linear unit activations. *SIAM Journal on Mathematics of Data Science* 2(4):1004–1033.

Wang, Hongyi, Saurabh Agarwal, and Dimitris Papailiopoulos. 2021a. Pufferfish: Communication-efficient models at no extra cost. *Proceedings of Machine Learning and Systems* 3:365–386.

———. 2021b. Pufferfish: Communication-efficient models at no extra cost.

Wang, Hongyi, Kartik Sreenivasan, Shashank Rajput, Harit Vishwakarma, Saurabh Agarwal, Jy-yong Sohn, Kangwook Lee, and Dimitris Papailiopoulos. 2020. Attack of the tails: Yes, you really can backdoor federated learning. *Advances in neural information processing systems*.

Wang, Yulong, Xiaolu Zhang, Lingxi Xie, Jun Zhou, Hang Su, Bo Zhang, and Xiaolin Hu. 2019. Pruning from Scratch. *arXiv:1909.12579 [cs]*.

Wei, Colin, Yining Chen, and Tengyu Ma. 2022a. Statistically meaningful approximation: a case study on approximating turing machines with transformers. *Advances on Neural Information Processing Systems (NeurIPS)*.

Wei, Jason, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. 2022b. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*.

Wei, Jason, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Chi, Quoc Le, and Denny Zhou. 2022c. Chain of thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*.

Weiss, Gail, Yoav Goldberg, and Eran Yahav. 2021. Thinking like transformers. In *International conference on machine learning*, 11080–11090. PMLR.

Wen, Wei, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. 2016. Learning structured sparsity in deep neural networks. *arXiv preprint arXiv:1608.03665*.

Wiesler, Simon, and Hermann Ney. 2011. A convergence analysis of log-linear training. *Advances in Neural Information Processing Systems* 24.

Woodworth, Blake, Brian Bullins, Ohad Shamir, and Nathan Srebro. 2021. The min-max complexity of distributed stochastic convex optimization with intermittent communication. *arXiv preprint arXiv:2102.01583*.

Woodworth, Blake, Kumar Kshitij Patel, Sebastian Stich, Zhen Dai, Brian Bullins, Brendan McMahan, Ohad Shamir, and Nathan Srebro. 2020a. Is local SGD better than minibatch SGD? In *International conference on machine learning*, 10334–10343. PMLR.

Woodworth, Blake E, Kumar Kshitij Patel, and Nati Srebro. 2020b. Minibatch vs local SGD for heterogeneous distributed learning. *Advances in Neural Information Processing Systems* 33:6281–6292.

Wright, Stephen J, and Ching-pei Lee. 2017. Analyzing random permutations for cyclic coordinate descent. *arXiv preprint arXiv:1706.00908*.

Wu, Jiaxiang, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. 2016. Quantized convolutional neural networks for mobile devices. In *Proceedings of the ieee conference on computer vision and pattern recognition*, 4820–4828.

Yan, Wilson, Yunzhi Zhang, Pieter Abbeel, and Aravind Srinivas. 2021. Videogpt: Video generation using vq-vae and transformers. *arXiv preprint arXiv:2104.10157*.

Yang, Greg, Jeffrey Pennington, Vinay Rao, Jascha Sohl-Dickstein, and Samuel S Schoenholz. 2019. A mean field theory of batch normalization. *arXiv preprint arXiv:1902.08129*.

Yehudai, Gilad, and Ohad Shamir. 2019. On the power and limitations of random features for understanding neural networks. *Advances in Neural Information Processing Systems* 32.

Ying, Bicheng, Kun Yuan, and Ali H Sayed. 2020. Variance-reduced stochastic learning under random reshuffling. *IEEE Transactions on Signal Processing* 68:1390–1408.

Ying, Bicheng, Kun Yuan, Stefan Vlaski, and Ali H Sayed. 2018. Stochastic learning under random reshuffling with constant step-sizes. *IEEE Transactions on Signal Processing* 67(2): 474–489.

Yosinski, Jason, Jeff Clune, Yoshua Bengio, and Hod Lipson. 2014. How transferable are features in deep neural networks? *Advances in neural information processing systems* 27.

Yu, Hao, Sen Yang, and Shenghuo Zhu. 2019. Parallel restarted sgd with faster convergence and less communication: Demystifying why model averaging works for deep learning. In *Proceedings of the aaai conference on artificial intelligence*, vol. 33, 5693–5700.

Yuan, Li, Yunpeng Chen, Tao Wang, Weihao Yu, Yujun Shi, Zi-Hang Jiang, Francis EH Tay, Jiashi Feng, and Shuicheng Yan. 2021. Tokens-to-token vit: Training vision transformers from scratch on imagenet. In *Proceedings of the ieee/cvf international conference on computer vision*, 558–567.

Yun, Chulhee, Srinadh Bhojanapalli, Ankit Singh Rawat, Sashank Reddi, and Sanjiv Kumar. 2019a. Are transformers universal approximators of sequence-to-sequence functions? In *International conference on learning representations*.

Yun, Chulhee, Shashank Rajput, and Suvrit Sra. 2022. Minibatch vs local sgd with shuffling: Tight convergence bounds and beyond. In *International conference on learning representations (iclr)*.

Yun, Chulhee, Suvrit Sra, and Ali Jadbabaie. 2019b. Small relu networks are powerful memorizers: a tight analysis of memorization capacity. In *Advances in neural information processing systems 32: Annual conference on neural information processing systems 2019, neurips 2019, december 8-14, 2019, vancouver, bc, canada*, 15532–15543.

———. 2021. Open problem: Can single-shuffle SGD be better than reshuffling SGD and GD? In *Conference on learning theory*, 4653–4658. PMLR.

Zaken, Elad Ben, Shauli Ravfogel, and Yoav Goldberg. 2021. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. *arXiv preprint arXiv:2106.10199*.

Zhang, Chiyuan, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. 2017a. Understanding deep learning requires rethinking generalization. In *International conference on learning representations*.

———. 2017b. Understanding deep learning requires rethinking generalization. In *5th international conference on learning representations, ICLR 2017, toulon, france, april 24-26, 2017, conference track proceedings*.

Zhou, Hattie, Janice Lan, Rosanne Liu, and Jason Yosinski. 2019. Deconstructing lottery tickets: Zeros, signs, and the supermask. *arXiv preprint arXiv:1905.01067*.

———. 2020. Deconstructing Lottery Tickets: Zeros, Signs, and the Supermask. *arXiv:1905.01067 [cs, stat]*.

Zhou, Hattie, Azade Nova, Hugo Larochelle, Aaron Courville, Behnam Neyshabur, and Hanie Sedghi. 2022. Teaching algorithmic reasoning via in-context learning. *arXiv preprint arXiv:2211.09066*.

Zhu, Chenzhuo, Song Han, Huizi Mao, and William J Dally. 2016. Trained ternary quantization. *arXiv preprint arXiv:1612.01064*.

Zhu, Michael, and Suyog Gupta. 2017. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878*.