ALGEBRAIC AND GEOMETRIC STRUCTURE IN MACHINE LEARNING AND OPTIMIZATION ALGORITHMS

By

Zachary Charles

A dissertation submitted in partial fulfillment of the

requirements for the degree of

Doctor of Philosophy

(Mathematics)

at the

UNIVERSITY OF WISCONSIN – MADISON

2017

Date of final oral examination: December, 2017

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

N. Boston, Professor, Mathematics, Electrical and Computer Engineering

B. Lesieutre, Professor, Electrical and Computer Engineering

D. Papailiopoulos, Assistant Professor, Electrical and Computer Engineering

R. Willett, Associate Professor, Electrical and Computer Engineering

Abstract

As the scope and importance of machine learning applications widens, it becomes increasingly important to design machine learning and optimization methods that will efficiently and provably obtain desired outputs. We may wish to guarantee that an optimization algorithm quickly converges to a global minimum or that a machine learning model generalizes well to unseen data. We would like ways to better understand how to design and analyze such algorithms so that we can make such guarantees.

In this thesis, we take an algebraic and geometric approach towards understanding machine learning and optimization algorithms. We show that various problems in both areas have algebraic or geometric structure and that we can use this structure to design more accurate and efficient algorithms. This geometric and algebraic viewpoint allows us to design improved optimization methods for a control–theoretic problem, address structured clustering problems even in the presence of missing data, bound the generalization error of learned models, and design more robust distributed optimization algorithms.

The first part of this thesis addresses the Belgian Chocolate problem, an open problem bridging optimization and control theory. We show that this problem has natural algebraic structure that arises from the theory of stable polynomials. We use this structure to design an algebraic optimization method for this problem. This method allows us to find larger values of the optimization version of the Belgian Chocolate problem than have previously been found. It is also more efficient than previous approaches, and allows us to tackle higher-order settings that have been previously out of reach.

In the second part, we analyze subspace clustering, the process of finding a union

of subspaces that best fit a collection of data points. We describe an algorithm for solving this problem. This algorithm naturally generalizes one popular method, sparse subspace clustering (SSC) [31]. Using the underlying geometry of these subspaces and our algorithm, we show that these algorithms can provably cluster the data correctly, even under the presence of noise or missing data. We do so by using the underlying convex geometry of the data points. In essence, we can reduce correctness of our algorithm to geometric properties of the data, and then show that with high probability these geometric conditions hold in certain models.

The third part of this thesis establishes novel generalization bounds for learning algorithms that converge to global minima. We show these bounds for non-convex loss functions satisfying the certain geometric conditions on their structure. We further show that these conditions arise for some neural networks with linear activations. Finally, we show that although stochastic gradient descent (SGD) and gradient descent generalize similarly in such settings, there exist simple neural networks with multiple local minima where SGD generalizes well but gradient descent does not.

In the final part, we develop a framework for deploying first-order optimization methods in distributed systems. Such algorithms are often beset by the straggler effect, where the slowest compute nodes in the system dictate the overall running time. In this work, we present computationally simple methods of assigning tasks in such systems that guarantee fast and approximately accurate distributed computation. By examining the problem from a linear algebraic point of view, we show that sacrificing a small amount of accuracy can significantly increase algorithmic robustness to stragglers.

Acknowledgements

I would like to thank my parents Rose and Cleve for their love and support. You taught me how to work hard in order to achieve my goals. Without your love, care, and frankly, patience for my own occasional ineptitude, none of this would have been possible.

Thank you to all my siblings, Shani, Mikki, Sara, Anthony, and Russell. I am a better person for growing up with you, learning from you, and getting to be a part of your lives. You are all wonderful and I love you dearly. Thank you in particular to Anthony for commiserating with me as a fellow graduate student. You inspire me to be curious, passionate, and hard-working. Thank you also to my siblings-in-law, Becky and Emily. I could not be happier to have you as part of my family and as part of those who have supported me through everything.

Thank you to my significant other, Kelda Baljon. You are a constant source of warmth, guidance, and goofy humor. I hope that this work can display even a fraction of the positivity and creativity you possess.

Thank you to my advisor, Nigel Boston for his guidance and advice over the years. I have thoroughly enjoyed the academic adventure I have been on over the last five years and I owe it in large part to your help and encouragement.

Thank you to Dimitris Papailiopoulos, Rebecca Willett, Bernie Lesieutre for not only being on my committee but for all your advice, support, your work with me, and for all you have taught me.

Thank you to my many other collaborators, mentors, and to other faculty and graduate students who have helped me get to where I am today. This (probably partial) list

includes Amin Jalali, Lalit Jain, Eric Bach, Jordan Ellenberg, Bob Barmish, Michael Ferris, Stephen Wright, and Dan Wu. I have been fortunate to learn from you all.

Thank you to my roommate, friend, and brilliant collaborator, Alisha Zachariah. All of this would have been so much harder without your friendship.

Finally, thank you to all of my other wonderful friends, including Micky Steinberg, Jason Steinberg, Brandon Alberts, Adrian Tovar Lopez, Iván Ongay-Valverde, Karla Ortiz, Ben Mark, Moisés Herradón, Eva Elduque, Solly Parenti, Juliette Bruce, Chris Janjigian, Jim Brunner, Nathan Clement, Daniel Hast, Soumya Sankar, Aman Abhishek, Dima Kuzmenko, Wanlin Li, and many others. You have made my time as a graduate student wonderful, weird, adventurous, and downright hysterical.

List of Figures

1 A convex shape (left) and a non-convex shape (right) in 2 dimensions. If we connect any two points in the shape on the left by a line, the line does not leave the shape. In the shape on the right, the line connecting two of the points leaves the shape. 26

2 The roots of the x, y, z corresponding to the largest δ found for $\deg(x) = 6$ in [20]. 34

3 The roots of the x, y, z corresponding to the largest δ found for $\deg(x) = 8$ in [20]. 35

4 The roots of the x, y, z corresponding to the largest δ found for $\deg(x) = 10$ in [20]. 36

5 The largest known quasi-admissible δ for x, y, z designed algebraically, for varying degrees of x. 50

6 A labeled dataset of cats (upper-left) and dogs (upper-right). In a supervised learning task, we may wish to predict the label of the image below. 63

7 An unlabeled dataset of cats and dogs plotted in 2 dimensions. We also give hypothetical clusters of this data set, denoted by various colors. Note that from these colors, we can see more easily which are cats and which are dogs. 64

8 An example of clustering points lying on 2-dimensional lines. On the left, we show the groupings obtained by standard clustering algorithms. In the middle, we show the underlying lines, and on the right we show the true clusters we wish to find. 65

9 A matrix representing hypothetical user-movie preferences. A 1 indicates the user rated a given movie positively, a 0 indicates the user rated a given movie negatively, and a ? means that the user has not rated that movie 66

10 The dual direction $v(x_i^{(\ell)}, X_{-i}^{(\ell)}, \lambda)$, where $x_i^{(\ell)}$ is a corrupted version of the true observation $y_i^{(\ell)}$. 82

11 The subspace incoherence μ_ℓ is the radius of the smallest sphere in the span of $X^{(\ell)}$ containing all projections of $y \in \mathcal{Y} \backslash \mathcal{Y}^{(\ell)}$ onto the polytope determined by the dual directions. 82

12 Convergence rates for T iterations of various gradient-based algorithms in the λ-SC and μ-PL settings. 157

13 The number of iterations T that achieves stability $\epsilon_{stab} = O(L^2/\mu n)$ for various gradient-based algorithms with step-size γ in the λ-SC and μ-PL settings. 158

14 A neural network representing a generalized quadratic model. 163

15 Graphs of the functions $\ell(w;(-1,1))$ (left), $\ell(w;(\frac{-1}{2},1))$ (middle), and $g(w) = \frac{1}{2}[\ell(w;(-1,1)) + \ell(w;(\frac{-1}{2},1))]$ (right). 164

16 Graph of the function $\ell(w;(-\frac{1}{2\hat{w}}),0)$. By construction, this function has critical points at $w = 0, \hat{w}, 2\hat{w}$. 164

18 A master-worker with three compute nodes. 173

19 A master-worker system with 4 workers and 1 gradient per worker. . . . 175

20 A master-worker system with 4 workers and 2 gradients per worker. . . . 176

21 A master-worker architecture of distributed computation with multiple cores per machines. Each of the k functions is assigned to a subset of n compute nodes. 180

22 A plot of the average one-step error $\text{err}_1(\mathbf{A})/k$ over 5000 trials. We take $k = 100$, $r = (1-\delta)k$ for varying δ. The figure on the left has $s = 5$ while the figure on the right has $s = 10$. 213

23 A plot of the average optimal decoding error $\text{err}(\mathbf{A})/k$ over 5000 trials. We take $k = 100$, $r = (1-\delta)k$ for varying δ. The figure on the left has $s = 5$ while the figure on the right has $s = 10$. 214

24 The average value of $\|\mathbf{u}_t\|_2^2/k$ of a BGC for $\delta \in \{0.1, 0.2, 0.3, 0.5, 0.8\}$ and varying t for 5000 trials. The figure on the left plots the algorithmic error for sparsity $s = 5$, while the figure on the right plots the algorithmic error for sparsity $s = 10$. 215

List of Recipes

1 Pie Crust 1
2 Yellow Cake 3
3 Rhubarb Upside-Down Cake 7
4 Fresh Pasta Dough 9
5 Chocolate Mousse Cake 24
6 Chocolate Olive Oil Cake 42
7 Chocolate Raspberry Cake 58
8 Croque-Madame 61
9 Gougères 78
10 Cranberry Pie with Streusel Topping 115
11 Blackberry Cheesecake Galette 127
12 Ginger Peach Pie 154
13 Hearty Wheat Bread 170
14 Challah with Raisins 188

Contents

Abstract **i**

Acknowledgements **iii**

1 Introduction **1**

1.1 How to Read this Thesis 2

1.2 Baking a Better Cake: Machine Learning and Optimization 3

1.3 The Science of Food v. the Role of Mathematics 6

2 Mathematical Background **9**

2.1 Optimization 10

2.1.1 Convexity 10

2.1.2 First-order Methods 13

2.2 Machine Learning 15

2.2.1 Optimization for Machine Learning 18

2.3 Summary of Results 21

I Algebraic Optimization for Simultaneous Stabilization **23**

3 Control, Optimization, and the Belgian Chocolate Problem **24**

3.1 Background 25
3.2 The Belgian Chocolate Problem 27
3.2.1 Our Contributions 29
3.3 Stability of Closed-Feedback Systems 31
3.4 Motivation for our approach 34
3.5 Admissible and Quasi-admissible δ 37
3.6 Mathematical Perspective and Main Results 39

4 Algebraic Optimization for the Belgian Chocolate Problem 42
4.1 Low degree examples 43
4.2 Algebraic specification 46
4.3 Approximating quasi-admissible δ by admissible δ 51
4.4 Optimality of algebraic specification 55

II Subspace Clustering 60

5 The Subspace Clustering Problem 61
5.1 Background 62
5.1.1 Clustering 62
5.1.2 Matrix Completion 65
5.2 Prior Work 67
5.2.1 Our Contributions 69
5.3 Problem Statement 70
5.3.1 Sparse Subspace Clustering 70
5.3.2 LS-SSC 72

5.4 Mathematical Perspectives and Summary of Results 76

6 Subspace Clustering with Missing and Corrupted Data 78

6.1 Preliminaries 79

6.1.1 Dual Programs and Convex Geometry 79

6.1.2 Dual Directions and Incoherence 81

6.2 Main Results 82

6.2.1 Deterministic Model 82

6.2.2 Random Model 84

6.2.3 Missing Data Model 85

6.3 Dual Certificates and the Deterministic Model 87

6.3.1 Dual Certificates 87

6.3.2 Bounding $\|\nu\|_2$ 92

6.3.3 Towards a Deterministic Criteria 100

6.3.4 Finding Admissible λ 101

6.3.5 Proof of Theorem 6.6 105

6.4 High-Dimensional Probability and the Random Model 106

6.5 Random Projections and Missing Data 111

III Stability and Generalization 114

7 Stability and Generalization of Learning Algorithms 115

7.1 Background 116

7.2 Prior work 118

7.2.1 Our Contributions 120

7.3 Algorithmic Stability 121
7.3.1 Stability and (Strongly) Convex Loss Functions 123
7.4 Mathematical Perspective and Main Results 124

8 Stability and the Polyak-Łojasiewicz Condition 127
8.1 The Polyak-Łojasiewicz and Quadratic Growth Conditions 128
8.2 Black-box Stability of Approximate Global Minima 133
8.2.1 Pointwise Hypothesis Stability for PL/QG Loss Functions 134
8.2.2 Uniform Stability for PL/QG Loss Functions 141
8.3 Examples of PL Loss Functions 146
8.3.1 Compositions of Strongly Convex and Piecewise-Linear Functions 146
8.3.2 Linear Neural Networks 148

9 Stability of Some First-order Methods 154
9.1 Stability for Strongly Convex and PL Loss Functions 155
9.2 Stability of Gradient Descent for Convex Loss Functions 157
9.3 Instability of Gradient Descent for Non-convex Loss Functions 162

IV Distributed Machine Learning and Gradient Coding 169

10 Gradient Coding 170
10.1 Background 171
10.1.1 Distributed Computation and the Straggler Effect 171
10.1.2 Distributed Machine Learning 174
10.2 Previous Work 177

10.2.1 Our Contributions 178
10.3 Problem Statement 179
10.3.1 Approximate Gradient Coding 181
10.4 Decoding 183
10.5 Mathematical Perspective and Main Results 185

11 Approximate Gradient Codes **188**
11.1 Fractional Repetition Codes 189
11.2 Adversarial Stragglers 194
11.2.1 Adversarial Stragglers and Fractional Repetition Codes 194
11.2.2 Adversarial Straggler Selection is NP-hard 196
11.3 Bernoulli Gradient Codes 200
11.3.1 Bounding the Decoding Error 201
11.3.2 One-step Error of Bernoulli Gradient Codes 204
11.3.3 Regularized Bernoulli Gradient Codes 209
11.4 Simulations 212
11.4.1 Decoding Error of Various Coding Schemes 212
11.4.2 Algorithmic Decoding Error of Bernoulli Coding 214

Bibliography **216**

Chapter 1

Introduction

Recipe 1: Pie Crust

Ingredients

- 2 $\frac{1}{2}$ cups (315 grams) flour
- 1 tablespoon (15 grams) sugar
- 1 teaspoon (5 grams) salt
- 1 cup (2 sticks) butter, cold
- 1 cup ice water
- (Optional but recommended) $\frac{1}{4}$ cup vodka, cold

Preparation

1. Dice butter and chill.
2. Combine flour, sugar, salt, and butter in a food processor. Process until the butter forms pea-sized chunks. Transfer to mixing bowl.
3. (Optional) If using vodka, sprinkle over butter mixture.
4. Add water 1 tablespoon at a time until the dough comes together.
5. Divide dough in half. Cover halves with plastic wrap and refrigerate for at least one hour, preferably longer.

I tried to come up with a "recipe" for a thesis for longer than I want to admit. The plan was to highlight the arc of my graduate studies. It would have included ingredients such as wonderful collaborators, the ability to accept failure, and coffee. While I still believe that there is a way to do this tastefully (no pun intended), I have not been able to find it. Instead, I included a recipe for pie crust from scratch. I hope that it serves the same function.[1]

1.1 How to Read this Thesis

There may be no correct way to read this thesis, but there is certainly an incorrect way to read it. Do not pass go, do not collect 200 dollars, and do not read this from beginning to end.

This work is divided into four parts, each dealing with a different topic somehow connected to machine learning and optimization. While related, none are necessary to understand each other. All four parts have self-contained, non-mathematical introductions in their first chapter. These are the appetizers, but they are designed to be a meal unto themselves. They explain what I have thought about for years and why that might have been an academically acceptable way to lead my life, if not socially acceptable.

Each introduction is followed by a mathematical background and a statement of results. These are perhaps the main courses, or better yet, today's menu. Any subsequent chapters are the all-too-rich desserts that should probably be refused for reasons of politeness and physical well-being.

[1] I am acutely aware that there is nothing new under the sun. This idea has been used to much better effect in works such as *Like Water for Chocolate* by Laura Esquivel and *Heartburn* by Nora Ephron. I can only hope that this work is at least unique in being the first dissertation on machine learning to reference these novels.

The soup du jour happens to be this chapter. I strongly recommend it, especially with a crusty slice of bread. If I had to recommend further reading to see if the taste is acceptable, I would do so as follows:

- If you are in the mood for chocolate, see Part I.
- If you are partial to traditional French cooking, sample Part II.
- If the pie crust recipe has piqued your interest, try Part III.
- If working with yeast is not too intimidating, check out Part IV.

1.2 Baking a Better Cake: Machine Learning and Optimization

The following is a simple, popular recipe for yellow cake. It has never produced a cake with which I have been happy. Machine learning and optimization work in the same way.[2]

Recipe 2: Yellow Cake

Ingredients

- 4 cups (500 grams) cake flour
- 2 teaspoons (10 grams) baking powder
- 1 $\frac{1}{2}$ teaspoons baking soda
- 1 teaspoon salt
- 2 sticks (1 cup) butter
- 2 cups (400 grams) sugar
- 2 teaspoons (10 ml) vanilla extract

[2] I am of course taking some liberties here. Machine learning typically requires less butter.

- 4 large eggs, room temperature
- 2 cups buttermilk (475 ml), shaken

Preparation

1. Preheat oven to 350° F. Butter two 9-inch round cake pans and line with parchment paper.
2. In a large bowl, beat butter and sugar with a mixer until pale and fluffy. Beat in vanilla, followed by the eggs, 1 at a time. At low speed, beat in the buttermilk until just combined
3. Sift dry ingredients together in a medium bowl. Add this mixture to the wet ingredients in three stages, mixing until just incorporated.
4. Spread batter evenly in cake pans, then drop on to counter several times to eliminate air bubbles.
5. Bake for 35 to 40 minutes or until a toothpick inserted into the center of the cake comes out clean.
6. Frost to taste.

If you wanted to bake a cake, you could simply follow this recipe as is, with whatever technical abilities you currently possess. The end result would almost surely be worse than if the cake were made by someone using their own recipe they developed over time, with experimentation, technical know-how, and knowledge of how ingredients interact with each other.

The same is true of machine learning. Suppose we want to train a machine to accomplish task X. Instead of deciding on a recipe to follow, we set up a *loss function* that measures how well the machine performed on task X. Instead of following the preparation steps in the recipe, we use *optimization* to find a way for the machine to accomplish X that makes the loss function as small as possible. While it is usually simple to set up *some* loss function and try a standard out-of-the-box optimization method, this

often takes too much computation time or else results in a machine learning algorithm that is horribly inaccurate.

When designing a recipe, one has to be aware of what they're trying to accomplish, and how reasonable that goal is. How do we ensure the cake retains its moisture? Will the cakes be sturdy enough to withstand the ordeal of being frosted? Can the recipe be made by professional baker? What about an amateur baker, or a complete novice? We must ask the same questions of our machine learning setup.[3] Does the loss function capture what we want the machine to do? Can it be efficiently computed? How do we ensure that minimizing this loss actually corresponds to accomplishing X in practice?

Next comes the actual baking of the cake. What pieces of equipment do we need to do justice to the recipe? Are there any unspoken tricks that we need to know to make the cake light and fluffy? Can we perform step 3 in a slightly more efficient way (by adding all the dry ingredients at once) and still make a good cake? These are analogues of important questions in optimization. Will our optimization method lead to a good solution? Does the optimization require more computation power than exists at a research university? Will slight changes in our setup lead to drastic changes in its output?[4]

Whether baking a cake or teaching a machine to accomplish a task or optimizing a particularly important function, the questions are the same. For all three, we need ways of analyzing what goes into these methods and ways of understanding what will come out.

[3]Again, I am glossing over some details. As far as I am aware, frosting has yet to be useful in a machine learning context.

[4]This last question is pertinent to baking, machine learning, and optimization. It is also more relevant to writing this dissertation than I would like to admit.

1.3 The Science of Food v. the Role of Mathematics

Cooking, arguably much like machine learning, is mainly done heuristically. When cooking dinner, most people do not stop to check the sodium content of their soy sauce to see if it will change the texture of their stir-fry. After learning how to make a dish once or twice, we may occasionally glance at a recipe for reference, but we typically rely on intuition, experience, and improvisation. We substitute ingredients, we take shortcuts, we use a smaller pan than we really should because the larger one is dirty. We very rarely try to understand the chemistry and physics of water retention in the food currently cooking on the stove. For most, it is simply not worth the time and effort. Our weeknight meals are not gastronomically important enough to warrant this behavior.

At the highest levels of cooking, however, a deep knowledge of the ingredients and their chemical interactions is vital.[5] This knowledge can be built up using chemistry and physics, or it can be gained through experience over time. By the same token, machine learning and optimization knowledge can be built up academically, or it can be gained through experience in tuning algorithms. Ideally both, though machine learning in practice currently seems to rely on an enormous amount of intuition.

To me, mathematics plays the role of food science in machine learning and optimization. The discovery that vodka can be used to make flaky pie crust was aided by a sophisticated understanding of chemistry [62]. By the same token, the insight in how to effectively train neural networks was aided by knowledge of calculus and analysis [81]. With the advent of self-driving cars and medical diagnosis via machine learning, the

[5] If you are currently in the process of making meringues or the chocolate mousse cake in Recipe 5, remember to use a metal spoon when scooping egg whites, otherwise they will collapse.

ability to understand the algorithms we use will only become more important. This is what mathematics does for us. How do we ensure that self-driving cars will not learn dangerous driving behavior over time? How do we ensure that our machine will correctly diagnose patients with rare or even previously unknown conditions? How do we bring out the moisture in our cake in a technically simple way?

I cannot answer the first of these two questions, but I am confident that the answers will come in time using mathematics. As for the third, below is a variation on the traditional sour cream cake that gives a wonderfully moist lemon-rhubarb taste, topped with caramelized brown sugar.

Recipe 3: Rhubarb Upside-Down Cake

Ingredients

- $2\ \frac{1}{2}$ sticks ($1\ \frac{1}{4}$ cups) butter
- $1\ \frac{1}{2}$ pounds rhubarb, sliced into $\frac{1}{2}$-inch cubes
- 2 teaspoons cornstarch
- $1\ \frac{1}{2}$ cups sugar
- $\frac{1}{2}$ cup brown sugar
- $1\ \frac{1}{4}$ teaspoons baking powder
- $\frac{1}{2}$ teaspoon salt
- Zest of 1 lemon
- 1 teaspoon vanilla extract
- 4 large eggs, room temperature
- $\frac{1}{3}$ cup sour cream
- 2 teaspoons lemon juice

Preparation

1. Preheat oven to 325° F. Grease a 9-inch springform pan and line with parchment paper. Wrap bottom with foil and place on a baking sheet.
2. In a medium bowl, mix rhubarb, cornstarch, and $\frac{1}{2}$ cup sugar.

3. Mix the brown sugar and $\frac{1}{2}$ stick butter in a pan over medium heat. Whisk until smooth and bubbling. Sift together the cake flour, baking powder, and salt.

4. Using fingers, work remaining sugar and lemon zest together until uniform in color. Mix 2 sticks of butter with the sugar mixture until light and fluffy.

5. Add vanilla and mix well. Add eggs, one at a time, mixing thoroughly after each addition. Mix in the sour cream and lemon juice.

6. Add the flour mixture in 4 stages, mixing each time until just incorporated.

7. Pour the brown sugar mixture into the cake pan, and spoon rhubarb and collected juices on top. Spoon in batter and smooth out.

8. Bake for 1 hour and 15 minutes, or until a toothpick stuck in the middle comes out clean.

Chapter 2

Mathematical Background

Recipe 4: Fresh Pasta Dough

Ingredients

- 2 cups flour
- 3 eggs
- 1 egg yolk

Preparation

1. Put flour in a large mixing bowl. Create a mound in the center.
2. Add eggs and yolk. Using a fork, beat the eggs and yolk and beginto incorporate the flour.
3. When the dough becomes pliable, use your fingers to mix the rest of the dough. Cover with plastic wrap and let rest for 15 minutes.
4. Cut and roll pasta by hand or with a pasta maker.

2.1 Optimization

Optimization involves minimizing or maximizing a function, possibly subject to one or more constraints. An *unconstrained optimization* problem is generally of the form

$$\min_x \quad f(x)$$

for some function f. Note that if we would like to maximize $f(x)$, we can instead minimize $-f(x)$. We will let f^* denote the minimum value of $f(x)$. Of course, most problems of interest, both theoretically and practically, involve constrains on what x are possible. To account for this, a general optimization problem is of the form

$$\begin{aligned} \min_x \quad & f(x) \\ \text{subject to} \quad & x \in \Omega. \end{aligned} \tag{2.1}$$

Here, Ω is our *feasible set*. Oftentimes Ω is constructed by requiring certain inequalities or equalities involving x to hold. The study of optimization generally involves understanding these problems and developing methods to solve them efficiently and accurately. As we will discuss below, optimization and machine learning are tightly intertwined, as most machine learning algorithms are trained via optimization problems.

While optimization of arbitrary functions over arbitrary domains remains a difficult task, these problems can become somewhat simpler when $f(x)$ and Ω are *convex*.

2.1.1 Convexity

In this section we will discuss convex sets and functions and their implications for optimization. For simplicity, we will let Ω be a subset of $\mathbb{R}^n$, though optimization can be performed over more general spaces.

Definition 2.1. *A set $\Omega \subseteq \mathbb{R}^n$ is convex if for all $x, y \in \Omega$ and $t \in [0, 1]$,*

$$(1 - t)x_1 + tx_2 \in \Omega.$$

Many results in optimization and analysis more generally involve examining the line between two feasible points. By imposing the condition that this line is contained within the feasible set, we can provide better analysis of machine learning methods. We can extend this notion to functions relatively simply.

Definition 2.2. *Let $\Omega \subseteq \mathbb{R}^n$ be convex. A function $f : \Omega \to \mathbb{R}$ is convex if for all $x_1, x_2 \in \Omega$ and $t \in [0, 1]$,*

$$f((1 - t)x_1 + tx_2) \leq (1 - t)f(x_1) + tf(x_2).$$

Some important examples of convex functions include affine functions, and ℓ_p norms for $p \geq 2$. More generally, any norm a on vector space is convex by the triangle inequality and homogeneity. These are especially important in machine learning as we often use specific norms in our loss functions in an attempt to make the output have some desirable property.

We say that f is *strictly convex* if the inequality above is replaced by a strict inequality. We can further restrict to a smaller, better behaved class of functions called *strongly convex* functions.

Definition 2.3. *Let $\Omega \subseteq \mathbb{R}^n$ be convex. A function $f : \Omega \to \mathbb{R}$ is strongly convex with parameter $\lambda > 0$ if $g(x) := f(x) - \frac{\lambda}{2}\|x\|_2^2$ is convex.*

We will often use the fact that if f is twice-differentiable, then λ-strong convexity is equivalent to the condition that for all $x, y \in \Omega$,

$$f(y) \geq f(x) + \langle f(x), y - x\rangle + \frac{\lambda}{2}\|y - x\|_2^2.$$

To see that convexity is important to optimization, note that we have the following lemma about the global minima of convex functions.

Lemma 2.4. *Suppose $f : \mathbb{R}^n \to \mathbb{R}$ is a differentiable, convex function. If x^* is a global minimizer of f then $\nabla f(x^*) = 0$.*

Suppose further that f is twice-differentiable. If x^* is a global minimizer of f, then $\nabla f(x) = 0$ and $\nabla^2 f(x^*)$ is positive semi-definite. Conversely, if $\nabla f(y) = 0$ and $\nabla^2 f(y)$ is positive-definite, then y is a global minimizer of f.

Points where $\nabla f(x) = 0$ are referred to as *critical points*. Convex functions give us useful geometric conditions on their global minima and in some cases, a useful test for global optimality. We will use such properties when analyzing widely-used optimization methods. We will also require the following notions.

Definition 2.5. *A function $f : \Omega \to \mathbb{R}$ is L-Lipschitz if for all $x_1, x_2 \in \Omega$,*

$$|f(x_1) - f(x_2)| \leq L\|x_1 - x_2\|.$$

If f is assumed to be differentiable, this is equivalent to saying that for all x, $\|\nabla f(x)\|_2 \leq L$. If the gradient of f is Lipschitz, then we say that the function is smooth, as in the following definition.

Definition 2.6. *A function $f : \Omega \to \mathbb{R}$ is β-smooth if for all $u, v \in \Omega$, we have*

$$\|\nabla f(u) - \nabla f(v)\|_2 \leq \beta\|u - v\|.$$

Note that the β-smooth condition implies that for all x, y,

$$f(y) \leq f(x) + \langle \nabla f(x), y - x \rangle + \frac{\beta}{2}\|y - x\|_2^2. \tag{2.2}$$

2.1.2 First-order Methods

To solve optimization problems, we often use *iterative*, *first-order* methods. In an iterative approach, we start off with some point x_0. At each step $t \geq 1$, we then apply some update rule $x_t = g(x_{t-1})$ in order to eventually get to a point x_T that is close to being a minima of f. A method is said to be first-order if it involves the function $f(x)$ that we are operating and its first derivative $\nabla f(x)$, but no higher-order derivatives.

The most famous iterative, first-order method is called gradient descent, is described below. Gradient descent typically involves fixing a number of iterations T and a step-size parameter γ.

Algorithm 1: Gradient Descent

Input: A number of iterations T and a step-size γ.

1. Set w_0 to be some initial feasible model.
2. For $t = 1, \ldots, T$, set $x_t = x_{t-1} - \gamma \nabla f(x_{t-1})$.
3. Output w_T.

The key insight of gradient descent is that by if we move in the negative gradient of a function, we should generally decrease the function value. By doing this for enough steps, we hope to eventually converge to a local or global minima of $f(x)$. Various adjustments can be made to this algorithm, such as changing the step-size at each step, but we will focus on the algorithm above for now. We will also note that the first step can be performed easily when we are performing unconstrained optimization, but may require more sophisticated methods, including interior-point methods [98] when Ω is an arbitrary convex set.

If f satisfies certain geometric properties, such as convexity, smoothness, and Lipschitz, we can give guarantees on the performance of gradient descent. This is an extremely pervasive and important theme throughout optimization and worth restating.

Key idea: By identifying the underlying structure of a function $f(x)$ and a feasible region Ω, we can hope to design better algorithms and provide better analyses for optimization.

As a simple example, when f is convex and L-Lipschitz we can show the following.

Lemma 2.7. *Suppose that $f(x)$ is a convex, L-Lipschitz and that we produce T iterations of gradient descent, initializing at a point x_0. Let $\Delta := f(x_0) - f^*$. If we choose a step-size of $\gamma = \Delta/L\sqrt{T}$, then*

$$f\left(\frac{1}{T}\sum_{t=0}^{T} x_t\right) - f^* \leq \frac{\Delta L}{\sqrt{T}}.$$

In other words, the average of all the iterations produced will have error that decreases with T. By taking T large enough, we can get arbitrarily close to f^*. Historically, convexity is an enormous workhorse in such proofs and allows us to get a better handle on the sub-optimality of the iterates. By contrast, much less can be said about the convergence of gradient descent for non-convex functions. Moreover, in the non-convex case we can only guarantee that gradient descent eventually converges to a critical point, not a global minimizer. For further reference on such topics, see [84].

If we further restrict to strongly convex and smooth functions, we can improve our convergence substantially. For details and further analysis, see [16].

Lemma 2.8. *Suppose that $f(x)$ is λ-strongly convex and β-smooth, and that we produce T iterations of gradient descent, initializing at a point x_0. Let $\Delta := f(x_0) - f^*$. If we*

choose a step-size of $\gamma = 1/\beta$*, then*

$$f(x_T) - f^* \leq \left(1 - \frac{\lambda}{\beta}\right)^T \Delta.$$

By restricting to convex functions, we can use first-order methods to solve optimization problems more efficiently than we can for arbitrary functions. There are many other methods to solve convex optimization problems, such as interior-point methods [98], but first-order methods end up being particularly important to machine learning. In particular, there are many useful variants of gradient descent that have strong theoretical and practical implications for machine learning. We discuss some below.

2.2 Machine Learning

Machine learning is growing in scope and ability at such a rapid rate that it is worth stepping back and formally defining the goals of machine learning and how they are accomplished. In the prototypical machine learning setting, we are given a large set of examples

$$S = \{(x_1, y_1), \ldots, (x_n, y_n)\}$$

where each vector $x_i \in \mathbb{R}^n$ is a *feature vector* and $y_i \in \mathbb{R}$ is its *label*. For instance, if we are trying to determine whether a given image has a dog, we could construct x as a vectorized version of the gray-scale image (so that it has real values), and its label could be 1 if the image contains a dog and -1 otherwise. We want to design a *prediction function* h that maps feature vectors to their correct label. Of course, we would like h to at the very least succeed on our training set S. Therefore, we would like h that

minimizes the *empirical risk* function

$$R_S[h] := \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}[h(x_i) \neq y_i]. \tag{2.3}$$

Here, $\mathbf{1}[A]$ is 1 if A is true and 0 otherwise. The links to optimization rear their head almost immediately under this framework. We want to somehow minimize $R_S[h]$ over the space of prediction functions h. In practice, simply minimizing $R_S[h]$ is not good enough, as R_S only contains information about data we have already seen. We would like to design h so that it also *generalizes* well to unseen data. We will discuss the question of how to guarantee that our classifiers generalize in Part III of this work.

This framework suggests, correctly, that we have an enormous amount of flexibility in designing h. To design h in practice, we often want a *model-based* classifier. In many scenarios, it is not good enough to have an accurate classifier. We would like the classifier to be based on some underlying model so that we can hope to understand the classifier, interpret its results in terms of the model, and compare it to other models. In such instances, our prediction function h is a function of the data point $x_i \in \mathbb{R}^n$ and a model $w \in \mathbb{R}^m$. For example, we could hope that the label y_i is a linear function of the feature vector x_i. That is, for some w^*,

$$y_i = x_i^T w^*.$$

Then, for a given w, our prediction would be $h(w; x_i) = x_i^T w$. The vector w allows us to better understand the prediction function, as it encodes which features of x_i are important.

One can think of this w as a representation of the classification function $h(w; \cdot)$. We will typically fix h and then identify the space of possible prediction functions with the space of possible models w. This is particularly useful in trying to minimize (2.3),

as it allows us to optimize over $\mathbb{R}^m$ instead of over the space of all possible functions. This allows us to utilize geometric notions and conditions from finite-dimensional vector space, and allows us to apply optimization algorithms such as gradient descent. While this representation is not strictly necessary for most of the following, we will take a model-based approach when possible due to its simplicity and widespread prevalence in modern machine learning theory and applications.

We would like to find a model w such that the function $h(w;\cdot)$ minimizes (2.3). Unfortunately, optimizing this function directly is difficult and expensive. This is due to the fact that $\mathbf{1}[h(w;x_i) - y_i]$ is not differentiable or even continuous. To rectify this, we will instead use more general *loss functions* to capture the difference between our predicted label and the true label.

For example, consider the situation that, given a model w, our predicted label is $\hat{y}_i = x_i^T w$. We wish to measure how close our predicted $\hat{y}_i$ comes to the true y_i. Since these are both real numbers, one natural approach is to simply look at $(\hat{y}_i - y_i)^2$. Note that this is 0 iff $\hat{y}_i = y_i$, and positive otherwise. Moreover, this quantity is smaller if our prediction is closer to y_i. Therefore, instead of using $\mathbf{1}[\hat{y}_i \neq y_i]$ in (2.3), we will define

$$\ell(w;(x_i,y_i)) = (x_i^T w - y_i)^2$$

and try to find a model w minimizing

$$\ell(w) := \frac{1}{n}\sum_{i=1}^{n} \ell(w;(x_i,y_i)). \tag{2.4}$$

As long as each $\ell(\cdot;(x_i,y_i))$ is differentiable and convex, the function ℓ will be as well.

In general, finding prediction functions to solve machine learning problems reduces to attempting to minimize a function of the form given in (2.4). The structure of the

loss function ℓ may differ from problem to problem. In order to ensure that w satisfies certain desirable properties, such as sparsity or constraints on how large w is in norm, we often add a *regularizer*. That is, we add some function $g(w)$ that penalizes w not satisfying certain properties. For example, we might use

$$\ell(w) = \frac{1}{n}\sum_{i=1}^{n} \ell(w; (x_i, y_i)) + \frac{\lambda}{2}\|w\|_2^2$$

to help contrain the size of w in the 2-norm, or we might use

$$\ell(w) = \frac{1}{n}\sum_{i=1}^{n} \ell(w; (x_i, y_i)) + \lambda\|w\|_1$$

to help encourage w to be sparse. Oftentimes we solve this problem for multiple $\lambda > 0$ and select the best one, or choose λ according to theoretical analysis. Note that regularizing by $\lambda g(w)$ can still be incorporated into the framework of (2.4) by taking the ith loss to be $\ell(w; (x_i, y_i)) + \frac{\lambda}{n} g(w)$.

2.2.1 Optimization for Machine Learning

As we discussed above, by using loss functions with desirable properties, we can turn the difficult optimization problem in (2.3) into (2.4), which can be much easier to optimize. In particular, if $\ell(\cdot; (x, y))$ is convex and differentiable, then $\ell(w)$ will be as well. We can then apply standard optimization techniques such as the aforementioned gradient descent to find an approximate minimizer. Unfortunately, the *large-scale* nature of machine learning problems may make gradient descent unreasonable to use in practice.

For simplicity of nation, let $\ell_i(w) := \ell(w; (x_i, y_i))$. Then note that standard gradient properties imply

$$\nabla\ell(w) = \frac{1}{n}\sum_{i=1}^{n} \nabla\ell_i(w).$$

In other words, to perform one iteration of gradient descent, we must compute the gradient of all n of our loss functions. Since we get one loss function per data point, this means that each step requires as many gradient computations as points in our data. Given that modern machine learning works with millions, even billions of data points, this may not be practical.

To get around this, we can use variants of gradient descent that involve less computation per iteration. The most famous such example is *stochastic gradient descent* (SGD). In SGD, we pick a random gradient to compute at each iteration instead of computing every gradient at each step. Just like gradient descent, SGD involves fixing a number of iterations T and a step-size γ_t for each iteration.

Algorithm 2: Stochastic Gradient Descent

Input: A number of iterations T and step-sizes γ_t for $1 \leq t \leq T$.

1. Set w_0 to be some initial feasible model.
2. For $t = 1, \ldots, T$ do:
   Select $i_t \in \{1, \ldots, n\}$ uniformly at random.
   Set $w_t = w_{t-1} - \gamma_t \nabla \ell_i(w_{t-1})$.
3. Output w_T.

Note that we now only need to perform one gradient computation per iteration. This generally comes at a trade-off of having to perform more iterations to achieve the same level of convergence as gradient descent. Still, this trade-off often works in favor of SGD, making it the de facto standard in machine learning applications.

As one might expect, geometric properties of the loss functions ℓ_i will allow us to control the convergence of SGD. There has been an enormous amount of work in the convergence properties of SGD over the last decade. We will present one such theorem here, but a more detailed collection of such results can be found in [16].

Lemma 2.9. *Suppose that each ℓ_i is λ-strongly convex and L-Lipschitz. If we run T iterations of SGD with step-sizes $\gamma_t = 2/\lambda(t+1)$ then the iterates satisfy*

$$f\left(\sum_{t=1}^{T} \frac{2t}{T(T+1)} w_t\right) - f^* \leq \frac{2L^2}{\lambda(T+1)}.$$

In general, this theorem tells us that as long as we run T iterations with a specific step-size, then our error decreases like $\frac{1}{T}$.

We can also interpolate between SGD and gradient descent using *mini-batch* SGD. In this algorithm, we pick a random batch of examples and compute their associated gradients. A full description of the algorithm is given below.

Algorithm 3: Mini-batch Stochastic Gradient Descent

Input: A number of iterations T, a batch-size B, and step-sizes γ_t for $1 \leq t \leq T$.

1. Set w_0 to be some initial feasible model.
2. For $t = 1, \ldots, T$ do:
   Select a set S_t of B training examples uniformly at random.
   Set
   $$w_t = w_{t-1} - \frac{\gamma_t}{B} \sum_{i \in S_t} \nabla \ell_i(w_{t-1}).$$
3. Output w_T.

By setting B larger than 1 but smaller than n, we hope to make each iteration relatively inexpensive, while requiring relatively few iterations for convergence. Many machine learning algorithms rely heavily on analyses of SGD and gradient descent, as well as computational tools for computing sums of gradients quickly. Doing such gradient computations in a distributed setting will be discussed below in Part IV.

Many other variants of gradient descent have been proposed, especially for training machine learning models, that also balance computational complexity per iteration and the number of iterations required for convergence. Such algorithms include the stochastic variance-reduced gradient method (SVRG) [46] and randomized coordinate descent

(RCD) [68].

In general, much of the work bridging machine learning and optimization involves designing loss functions and regularization functions that accurately describe a problem, and then determining how to best use optimization methods to find models minimizing those loss functions. Further work must then be put into guaranteeing that the output of these algorithms possesses properties that are desirable, or even necessary, for the machine learning application.

2.3 Summary of Results

This thesis will attempt answer various questions lying at the intersection of optimization and machine learning. The moral of these results is that a better understanding of algebraic and geometric structure can lead to better optimization and machine learning algorithms. In particular, this geometric and algebraic structure allows us to design more efficient algorithms for machine learning problems and better understand the structure of the output of said algorithms. This allows us to prove that our algorithms give outputs that are not only "correct" from an optimization point of view, but also useful in machine learning contexts.

In Part I, we will analyze a non-convex optimization problem motivated by control theory. Despite the general difficulty of non-convex optimization, we will show that we can use underlying algebraic structure to design a more accurate and efficient optimization method for solving the problem. In Part II, we will consider the problem of subspace clustering and design convex optimization methods for the problem. Using linear algebraic and geometric structure of the problem, we will prove that the output

of these programs can be used to solve the overarching problem. In Part III, we will consider the question of how to guarantee that the output of optimization problems in machine learning generalize to unseen data. We will show that by assuming general geometric conditions on the loss functions, we can give strong guarantees on the generalization of our machine learning models. Finally, in Part IV, we will discuss how to use first-order methods for machine learning in a distributed setting. We do so by taking a linear algebraic view of ways to assign tasks in distributed systems. Our main results show that by allowing for small amounts of error in gradient computations, we can make our distributed algorithms much more efficient.

Part I

Algebraic Optimization for Simultaneous Stabilization

Chapter 3

Control, Optimization, and the Belgian Chocolate Problem

Recipe 5: Chocolate Mousse Cake

Ingredients

- 10 ounces bittersweet chocolate
- 9 tablespoons butter
- 6 large eggs, separated
- Pinch of salt
- $\frac{3}{4}$ cup sugar
- (Optional) 2 tablespoons brandy
- 1 teaspoon confectioner's sugar

Preparation

1. Preheat the oven to 375° F, and position oven rack in lower center of the oven. Lightly grease a 9-inch springform pan and cover the bottom with foil. Place on top of baking sheet.

2. Using a double boiler, melt together chocolate and butter. Set aside to cool.

3. In a bowl, whisk egg yolks and $\frac{1}{2}$ cup sugar until pale and frothy. Whisk in brandy if using, then fold in chocolate mixture.

4. Using an electric mixer, whisk egg whites and salt until thick. Add remaining sugar and whisk until stiff.

Fold egg whites into chocolate mixture in 2 stages.

5. Pour mixture into pan and and bake for 15 minutes. Turn off oven and crack open oven door. Let cake remain in oven for 15 minutes.
6. Allow cake to cool completely. Just before serving, dust with confectioner's sugar.

In the following chapters, we will discuss the oddly named *Belgian chocolate problem*. Unfortunately, the problem does not actually involve any chocolate. To remedy this, I have provided recipes involving chocolate that can and should be eaten while perusing this part.

3.1 Background

For better or for worse, mathematical problems of practical interest can often be reduced to optimizing a certain quantity. In some settings, this optimization problem lies in plain sight, such as if we want to configure a quantity (such as price) in a way that maximizes profit. Other times, the underlying optimization problem is less obvious. For example, suppose we wish to design a machine learning algorithm that predicts whether a given image contains a dog. To do so, we typically try to find a *model* that minimize a specific *loss function*. The loss function quantifies how many images your model correctly predicts as containing a dog.

In optimization we are typically interested in finding the quantity x^* that maximizes (or minimizes) some function $f(x)$ over all possible x. The possible x lie in some region which we refer to as the *feasible region*. Not all x may be feasible. For example, if we

are setting a price of a good, then in many applications (but certainly not all) we might want to restrict to having positive prices. This alters the region of prices that we wish to consider. The difficulty of our optimization problem is affected by the geometry of the feasible region and the structure of the function $f(x)$.

One particularly desirable property of the feasible region is *convexity*. This property states that if we take any two feasible points and draw a line between them, then the line should not leave the region. Below are some examples of convex and non-convex shapes in 2 dimensions.

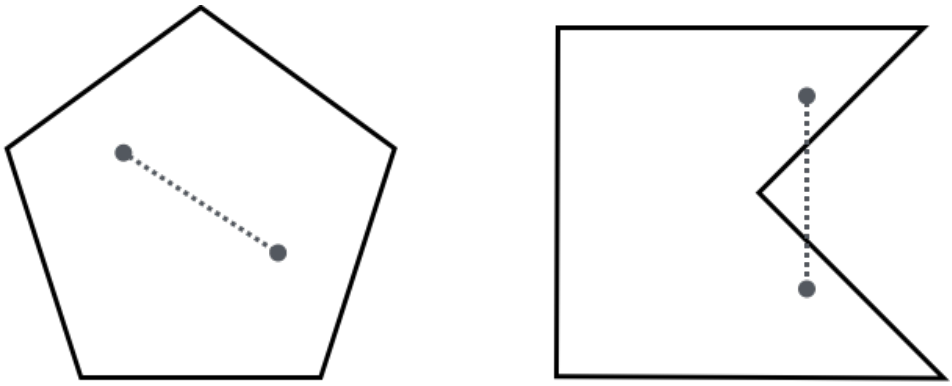


Figure 1: A convex shape (left) and a non-convex shape (right) in 2 dimensions. If we connect any two points in the shape on the left by a line, the line does not leave the shape. In the shape on the right, the line connecting two of the points leaves the shape.

This underlying geometry makes optimizing over convex feasible regions much simpler than optimizing over non-convex feasible region. Unfortunately, optimization problems of practical interest often have non-convex feasible regions. This means that standard optimization techniques are either ineffective or require you to run them for many more iterations than in the convex setting. To overcome these difficulties, we often have to use other structure contained in our problem.

In this chapter, we consider the Belgian chocolate problem,[1] a famous open problem bridging optimization and control theory. The problem involves maximizing a parameter δ over a complicated non-convex feasible region. We show that the problem has underlying *algebraic* structure. We show that by exploiting this structure, we can design better optimization methods for this problem. Moreover, our algebraic method does not require running the algorithm over and over to get incrementally larger and larger values. Instead of hoping that the algorithm eventually finds a good point, our method directly identifies points that provide the largest known value of the Belgian chocolate problem so far.

3.2 The Belgian Chocolate Problem

The Belgian chocolate problem is a famous open problem in control theory proposed by Blondel in 1994. In the language of control theory, Blondel wanted to determine the largest value of a process parameter for which stabilization of an unstable plant could be achieved by a stable minimum-phase controller [9]. Blondel designed the plant to be a low-degree system that was resistant to known stabilization methods, in the hope that a solution would lead to development of new stabilization techniques. Specifically, Blondel wanted to determine the largest value of $\delta > 0$ for which the transfer function $P(s) = (s^2 - 1)/(s^2 - 2\delta s + 1)$ can be stabilized by a proper, bistable controller.

For readers unfamiliar with control theory, this problem can be stated in simple algebraic terms. To do so, we will require the notion of a *stable* polynomial. A polynomial

[1] Mathematically, the problem has nothing to do with chocolate. Its name comes from the fact that Vincent Blondel, the one who proposed the problem, offered a kilogram of fine Belgian chocolate for its solution. While our work gets us closer to the chocolate than all previous attempts so far, we have not fully earned the kilogram of chocolate yet. One can dream.

is stable if all its roots have negative real part. The Belgian chocolate problem, then, is as follows.

Belgian chocolate problem: Determine for which $\delta > 0$ there exist real, stable polynomials $x(s), y(s), z(s)$ with $\deg(x) \geq \deg(y)$ satisfying

$$z(s) = (s^2 - 2\delta s + 1)x(s) + (s^2 - 1)y(s). \tag{3.1}$$

We call such δ *admissible*. In general, stability of x, y, z becomes harder to achieve the larger δ is. Therefore, we are primarily interested in the supremum of all admissible δ. If we fix a maximum degree n for x and y, then this gives us the following global optimization problem for each n.

Belgian chocolate problem (optimization version):

$$\begin{aligned} \underset{\delta, x(s), y(s)}{\text{maximize}} \quad & \delta \\ \text{subject to} \quad & x, y, z \text{ are stable}, \\ & z(s) = (s^2 - 2\delta s + 1)x(s) + (s^2 - 1)y(s), \\ & \deg(y) \leq \deg(x) \leq n. \end{aligned} \tag{3.2}$$

Note that we can view a degree n polynomial with real coefficients as a $(n+1)$-dimensional real vector of its coefficients. Under this viewpoint, the space of polynomials x, y, z that are stable and satisfy (3.1) is an extremely complicated non-convex space. As a result, it is difficult to employ global optimization methods directly to this problem. The formulation above does suggest an undercurrent of algebra in this problem. This will be exploited to transform the problem into a combinatorial optimization problem by finding points that are essentially local optima.

Previous work has employed various optimization methods to find even larger admissible δ. Patel et al. [72] were the first to show that $\delta = 0.9$ is admissible by x, y of degree at most 11, answering a long-standing question of Blondel. They further showed that $\delta = 0.93720712277$ is admissible. In 2005, Burke et al. showed that $\delta = 0.9$ is admissible with x, y of degree at most 3[17]. They also improved the record to $\delta = 0.94375$ using gradient sampling techniques. In 2007, Chang and Sahinidis used branch-and-reduce techniques to find admissible δ as large as 0.973974 [20]. In 2012, Boston used algebraic techniques to give examples of admissible δ up to 0.97646152 [11]. Boston found polynomials that are almost stable and satisfy (3.1). Boston then used ad hoc methods to perturb these to find stable x, y, z satisfying (3.1). While effective, no systematic method for perturbing these polynomials to find stable ones was given.

3.2.1 Our Contributions

In this chapter, we extend the approach used by Boston in 2012 [11] to achieve the largest known value of δ so far. We will refer to this method as the method of *algebraic specification*. We show that these almost stable polynomials serve as limiting values of the optimization program. Empirically, these almost stable polynomials achieve the supremum over all feasible δ. Furthermore, we give a theoretically rigorous method for perturbing the almost stable polynomials produced by algebraic specification to obtain stable polynomials. Our approach shows that all $\delta \leq 0.9808348$ are admissible. This gives the largest known admissible value of δ to date. We further show that previous global optimization methods are tending towards the limiting values of δ found via our optimization method.

We do not assume any familiarity on the reader's part with the algebra and control theory and will introduce all relevant notions. While we focus on the Belgian chocolate problem throughout the chapter, we emphasize that the general theme of this chapter concerns the underlying optimization program. We aim to illustrate that by considering the algebraic structure contained within an optimization problem, we can develop better global optimization methods.

While our approach may sound foreign, even to the reader versed in optimization, we will show that our algebraic optimization method outperforms prior global optimization methods for solving the Belgian chocolate problem. We will contrast our method with the optimization method of Chang and Sahinidis [20] in particular. Their method used iterative branch-and-reduce techniques [82] to find what was the largest known value of δ until our new approach. Due to the complicated feasible region, their method may take huge numbers of iterations or converge to suboptimal points. Our method eliminates the need for these expensive iterative computations by locating and jumping directly to the larger values of δ. This approach has two primary benefits over [20]. First, it allows us to more efficiently find δ as we can bypass the expensive iterative computations. This also allows us to extend our approach to cases that were not computationally tractable for [20]. Second, our approach allows us to produce larger values of δ by finding a finite set of structured limit points. In low-degree cases, this set provably contains the supremum of the problem, while in higher degree cases, the set contains larger values of δ than found in [20].

3.3 Stability of Closed-Feedback Systems

In control theory, we are often concerned with the stability of a dynamical system. We will focus on the setting that our dynamical system is a *closed-feedback system* (otherwise known as a *closed loop* system). In a closed-feedback system, the system adjusts its input based on its output. For notational purposes, given $t \in \mathbb{C}$, we let $\mathrm{Re}(t)$ denote its real part. We will let $\mathbb{R}[s]$ denote the set of polynomials in s with real coefficients.

As shown in most introductory references on control theory (see [5] for example), the dynamics of a single closed-feedback loop can be captured by its *transfer function*. In many typical applications, the this transfer function is a rational function. For the purposes of this work, we suppose that the transfer function $P(s)$ is rational and given by

$$P(s) = \frac{b(s)}{a(s)}$$

where a and b are coprime polynomials. We want to design a *controller* that somehow stabilizes the feedback system. Mathematically, a controller is a rational function

$$C(s) = \frac{y(s)}{x(s)}$$

where x and y are coprime polynomials. We are interested in the stability of the feedback system. In particular, we would like a system with bounded input to have bounded output (BIBO). As it turns out, this property is equivalent to the rational function being *stable*, as defined below.

Definition 3.1. *A rational function $\phi(s) : \mathbb{C} \to \mathbb{C}$ is stable if it has no poles $t \in \mathbb{C}$ with $Re(t) \geq 0$.*

In particular, $P(s) = b(s)/a(s)$ is stable iff $a(s)$ has no roots with nonnegative real part. We now define the notion of stability for a feedback system.

Definition 3.2. *A system with transfer function $P(s)$ and controller $C(s)$ is stable if the closed-loop transfer function*

$$H(s) := \frac{P(s)C(s)}{1 + P(s)C(s)}$$

is stable.

Using $P(s) = b(s)/a(s), C(s) = x(s)/y(s)$ for a, b coprime and x, y coprime, simple computations show the following lemma.

Lemma 3.3. *The system above is stable if and only if $a(s)x(s) + b(s)y(s)$ has no zero t such that $Re(t) \geq 0$.*

In control theory, we are often given $P(s)$ and asked to construct $C(s)$ so that the system is stable. Moreover, we may be interested in designing $C(s)$ so that it is stable or *bistable*, as defined below.

Definition 3.4. *A rational function $\phi(s) : \mathbb{C} \to \mathbb{C}$ is bistable if it has no poles or zeros with nonnegative real part.*

In other words, $\phi(s)$ is bistable iff both $\phi(s)$ and $\frac{1}{\phi(s)}$ are stable. In the Belgian chocolate problem, Blondel considered the transfer function

$$P(s) = \frac{s^2 - 1}{s^2 - 2\delta s + 1}.$$

Here, δ is a process parameter taking values between 0 and 1. Blondel was specifically interested in the following question concerning this transfer function.

Belgian chocolate problem: For which $\delta \in [0,1]$ can $P(s)$ be stabilized by a rational, bistable controller?

Let $C(s) = y(s)/x(s)$. Saying that $C(s)$ is bistable is equivalent to saying that x, y have no roots with nonnegative real part. In an unfortunate abuse of notation, we say that a polynomial is stable if it has no roots with nonnegative real part. In the following, this is the only notion of stability we will refer to. By Lemma , the Belgian chocolate problem is equivalent to the following question.

Belgian chocolate problem: For which $\delta \in [0,1]$ are there real, stable polynomials $x(s), y(s)$ such that

$$z(s) = (s^2 - 2\delta s + 1)x(s) + (s^2 - 1)y(s)$$

is also stable?

Equivalently, we want to design real, stable polynomials x, y, z satisfying

$$(s^2 - 2\delta s + 1)x(s) + (s^2 - 1)y(s) = z(s).$$

As it turns out, by relaxing the stability condition to one of quasi-stability (where we allow for roots with zero real part), we can construct such x, y, z relatively easily. We will then be able to approximate these quasi-stable polynomials by stable ones.

3.4 Motivation for our approach

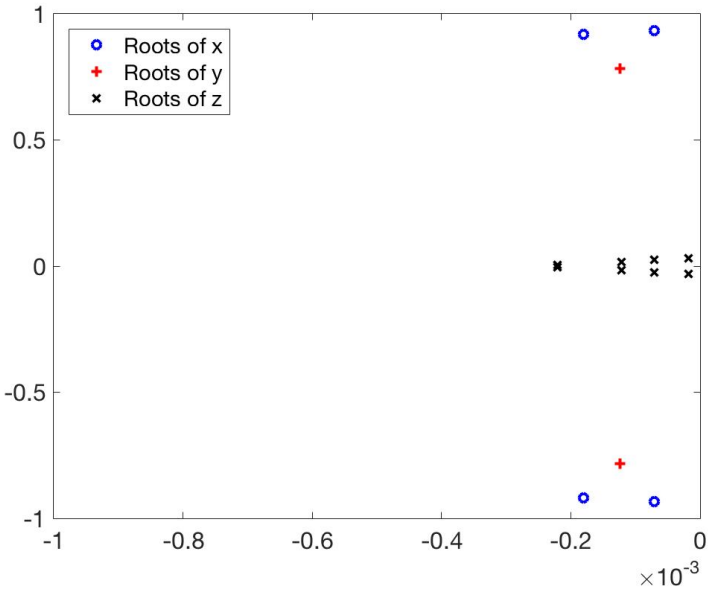


Figure 2: The roots of the x, y, z corresponding to the largest δ found for $\deg(x) = 6$ in [20].

In order to explain our approach, we will discuss previous approaches to the Belgian chocolate problem in more detail. Such approaches typically perform iterative non-convex optimization in the space of stable controllers in order to maximize δ. In [20], Chang and Sahinidis formulated, for each n, a non-convex optimization program that sought to maximize δ subject to the polynomials $x, y, (s^2 - 2\delta s + 1)x + (s^2 - 1)y$ being stable and such that $n \geq \deg(x) \geq \deg(y)$. For notational convenience, we will always define $z = (s^2 - 2\delta s + 1)x + (s^2 - 1)y$. Chang and Sahinidis used branch-and-reduce techniques to attack this problem for n up to 10. We plot these in Figures 2, 3, and 4.

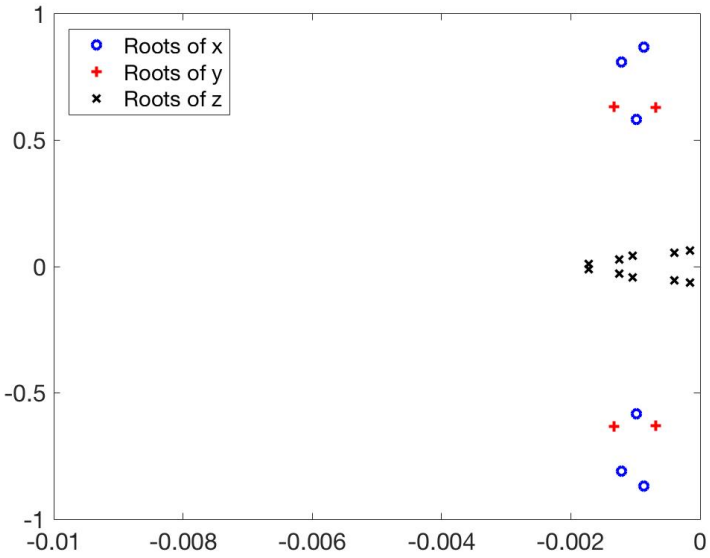


Figure 3: The roots of the x, y, z corresponding to the largest δ found for $\deg(x) = 8$ in [20].

We should note that these plots omit two roots of x in each case. The only roots of x that were omitted are very close to $-\delta \pm \sqrt{\delta^2 - 1}$. This suggests that x should have a factor close to $(s^2 + 2\delta s + 1)$. Examining the remaining roots, a pattern emerges. Almost all the roots of these polynomials are close to the imaginary axis and are close to a few other roots. In fact, most of these roots have real part in the interval $(-0.01, 0)$. In other words, the x, y, z are approximated by polynomials with many repeated roots on the imaginary axis.

This suggests the following approach. Instead of using non-convex optimization to iteratively push x, y, z towards polynomials possessing repeated roots on the imaginary axis, we will algebraically construct polynomials with this property. This will allow us to immediately find large limit points of the optimization problem in (3.2). While the x, y, z we construct are not stable, they are close to being stable. We will show later that

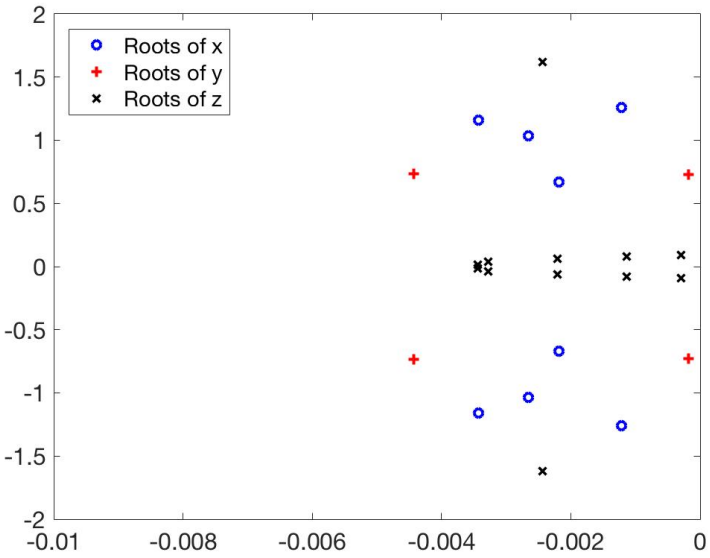


Figure 4: The roots of the x, y, z corresponding to the largest δ found for $\deg(x) = 10$ in [20].

we can perturb x, y, z and thereby push their roots just to the left of the imaginary axis, causing them to be stable. This occurs at the expense of decreasing δ by an arbitrarily small amount.

Our method only requires examining finitely many such limit points. Moreover, for reasonable degrees of x and y, these limit points can be found relatively efficiently. By simply checking each of these limit points, we reduce to a combinatorial optimization problem. This combinatorial optimization problem provably achieves the supremal values of δ for $\deg(x) \leq 4$. For higher degree x, our method finds larger values of δ than any previous optimization method thus far. In the sections below we will further explain and motivate our approach, and show how this leads to the largest admissible δ found up to this point.

3.5 Admissible and Quasi-admissible δ

In this section, we will relax the notion of stability to one of *quasi-stability*, and discuss this notion in the context of the Belgian Chocolate problem. Recall that for $p(s) \in \mathbb{R}[s]$, we call $p(s)$ *stable* if every root t of p satisfies $\mathrm{Re}(t) < 0$. Below, we define a relaxation of this condition where we allow roots to have zero real part.

Definition 3.5. *We say that a polynomial $p(s)$ is quasi-stable if every root t of p satisfies $Re(t) \leq 0$.*

We let H denote the set of all stable polynomials in $\mathbb{R}[s]$, and we let $\overline{H}$ denote the set of quasi-stable polynomials of $\mathbb{R}[s]$. Note that $H \subseteq \overline{H}$. We let $H^m, \overline{H^m}$ denote the sets of stable and quasi-stable polynomials respectively of degree at most m. These also satisfy $H^m \subseteq \overline{H^m}$. We will now define the notion of admissibility with respect to the Belgian chocolate problem.

Definition 3.6. *We call δ admissible if there exist $x, y \in H$ such that* $\deg(x) \geq \deg(y)$ *and*

$$(s^2 - 2\delta s + 1)x(s) + (s^2 - 1)y(s) \in H. \tag{3.3}$$

Definition 3.7. *We call δ quasi-admissible if there exist $x, y \in \overline{H}$ such that* $\deg(x) \geq \deg(y)$ *and*

$$(s^2 - 2\delta s + 1)x(s) + (s^2 - 1)y(s) \in \overline{H}. \tag{3.4}$$

Note that since quasi-stability is weaker than stability, quasi-admissibility is weaker than admissibility. Our main theorem (Theorem 3.10 below) will show that if δ is quasi-admissible, then all smaller δ are admissible. Note that this implies that the Belgian chocolate problem is equivalent to finding the supremum of all admissible δ. We will

then find quasi-admissible δ in order to establish which δ are admissible. This is the core of our approach. These quasi-admissible δ are easily identified and are limit points of admissible δ.

In practice, one verifies stability by using the Routh-Hurwitz criteria. Suppose we have a polynomial $p(s) = a_0 s^n + a_1 s^{n-1} + \ldots + a_{n-1} s + a_n \in \mathbb{R}[s]$ such that $a_0 > 0$. Then we define the $n \times n$ *Hurwitz matrix* $A(p)$ as

$$A(p) = \begin{pmatrix} a_1 & a_3 & a_5 & \ldots & \ldots & 0 & 0 \\ a_0 & a_2 & a_6 & \ldots & \ldots & 0 & 0 \\ 0 & a_1 & a_3 & \ldots & \ldots & 0 & 0 \\ 0 & a_0 & a_2 & \ldots & \ldots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \ldots & \ldots & a_{n-2} & a_n \end{pmatrix}.$$

Adolf Hurwitz showed that a real polynomial p with positive leading coefficient is stable if and only if all leading principal minors of $A(p)$ are positive. While it may seem natural to conjecture that p is quasi-stable if and only if all leading principal minors are nonnegative, this only works in one direction.

Lemma 3.8. *Suppose p is a real polynomial with positive leading coefficient. If p is quasi-stable then all the leading principal minors of $A(p)$ are nonnegative.*

Proof. If $p(s)$ is quasi-stable, then for all $\epsilon > 0$, $p(s + \epsilon)$ is stable. Therefore, for all $\epsilon > 0$, the leading minors of $A(p(s + \epsilon))$ are all positive. Note that

$$\lim_{\epsilon \to 0} A(p(s + \epsilon)) = A(p).$$

Since the minors of a matrix are expressible as polynomial functions of the entries of

the matrix, the leading principal minors of A are limits of positive real numbers. They are therefore nonnegative. □

To see that the converse does not hold, consider $p(s) = s^4 + 198s^2 + 101^2$. Its Hurwitz matrix has nonnegative leading principal minors, but p is not quasi-stable. This example, as well as a more complete characterization of quasi-stability given below, can be found in [4]. In particular, it is shown in [4] that a real polynomial p with positive leading coefficient is quasi-stable if and only if for all $\epsilon > 0$, $A(p(s + \epsilon))$ has positive leading principal minors.

3.6 Mathematical Perspective and Main Results

In general, we show that quasi-admissible δ can be used to find admissible δ. We first present the following theorem concerning which δ are admissible. We will defer the proof until later as it is a simple corollary to a stronger theorem about approximating polynomials in $\overline{H}$ by polynomials in H.

Theorem 3.9. *If δ is admissible then all $\hat{\delta} < \delta$ are also admissible.*

For $\delta = 1$, note that the Belgian chocolate problem reduces to whether there are $x, y \in H$ with $\deg(x) \geq \deg(y)$ such that $(s-1)^2 x + (s^2-1)y \in H$. This cannot occur for non-zero x, y since $(s-1)^2 x + (s^2-1)y$ has a root at $s = 1$. Theorem 3.9 then implies that any $\delta \geq 1$ is not admissible. In 2012, Bergweiler and Eremenko showed that any admissible δ must satisfy $\delta < 0.999579$ [8].

On the other hand, if we fix x, y then there is no single largest admissible δ associated to x, y. Standard results from control theory show that if δ is admissible by x, y then

for ϵ small enough, $\delta + \epsilon$ is admissible by the same polynomials.

Therefore, supremum δ^* over all admissible δ will not be associated to stable x, y. From an optimization point of view, the associated optimization program in (3.2) has an open feasible region. In particular, the set of admissible δ for (3.2) is of the form $(0, \delta_n^*)$ for some δ_n^* that is not admissible by x, y of degree at most n. However, as we will later demonstrate, quasi-admissible δ lie on the boundary of this feasible region. Moreover, quasi-admissible δ naturally serve as analogues of local maxima. We will therefore find quasi-admissible δ and use these to find admissible δ. In Section 4.3 we will prove the following theorem relating admissible and quasi-admissible δ. The following is the main theorem of our work and demonstrates the utility of searching for quasi-admissible δ.

Theorem 3.10. *If δ is quasi-admissible, then all $\hat{\delta} < \delta$ are admissible. Moreover, if δ is quasi-admissible by quasi-stable x, y of degree at most n, then any $\hat{\delta} < \delta$ is admissible by stable $\hat{x}, \hat{y}$ of degree at most n.*

This theorem shows that to find admissible δ, we need only to find quasi-admissible δ. In fact our theorem will show that if δ is quasi-admissible via x, y of degree at most n, then all $\hat{\delta} < \delta$ are admissible via x, y of degree at most n as well. In short, quasi-admissible δ serve as upper limit points of admissible δ. Also note that since admissible implies quasi-admissible, Theorem 3.10 implies Theorem 3.9.

The proof of Theorem 3.10 will be deferred until Section 4.3. In fact, we will do more than just prove the theorem. We will given an explicit algorithm for approximating quasi-stable $\hat{\delta}$ by stable δ within any desired tolerance. We will also be able to use the techniques in Section 4.3 to prove the following theorem showing that admissible δ are always smaller than some quasi-admissible δ.

Theorem 3.11. *If δ is admissible by x, y of degree at most n then there is some $\hat{\delta} > \delta$ that is quasi-admissible by $\hat{x}, \hat{y}$ of degree at most n. Moreover, this $\hat{\delta}$ is not admissible by these polynomials.*

In other words, for any admissible δ, there is a larger $\hat{\delta}$ that is quasi-admissible but not necessarily admissible. Therefore, we can restrict to looking at polynomials x, y, z with at least one root on the imaginary axis.

Chapter 4

Algebraic Optimization for the Belgian Chocolate Problem

Recipe 6: Chocolate Olive Oil Cake

Ingredients

- 1 $\frac{1}{2}$ cups flour
- $\frac{3}{4}$ cup cocoa powder
- 1 $\frac{1}{2}$ teaspoons baking soda
- $\frac{1}{2}$ teaspoon salt
- $\frac{3}{4}$ cup sugar
- $\frac{3}{4}$ cup brown sugar
- $\frac{1}{2}$ cup olive oil
- 1 $\frac{1}{2}$ cups coffee
- 1 tablespoon cider vinegar

Preparation

1. Heat oven to 350° F. Line a 9-inch round cake pan with parchment paper and grease.

2. Whisk flour, cocoa powder, baking soda, salt, and sugar in the bottom of a large mixing bowl. Add brown sugar and olive oil, whisking to combine. Add coffee and vinegar and whisk until smooth.

3. Pour into prepared pan. Bake for 30 to 35 minutes or until the top is springy and a tester inserted in the center comes out clean.

4. (Optional): Glaze with a frosting made from $\frac{3}{4}$ cup melted chocolate chips, 2 tablespoons cocoa powder, 3 table spoon olive oil, 2 tablespoons corn syrup, and a pinch of salt.

4.1 Low degree examples

In this section we demonstrate that in low-degree settings, the supremum of all admissible δ in (3.2) is actually a quasi-admissible δ. By looking at quasi-stable polynomials that are not stable, we can greatly reduce our search space and directly find the supremum of the optimization program in (3.2). For small degrees of x, y, we will algebraically design quasi-stable polynomials that achieve previously known bounds on the Belgian chocolate problem in these degrees.

Burke et al. [17] showed that for $x \in H^3, y \in H^0$, any admissible δ must satisfy $\delta < \sqrt{2+\sqrt{2}}/2$ and for $x \in H^4, y \in H^0$, δ must satisfy $\delta < \sqrt{10+2\sqrt{5}}/4$. He et al. [41] later found $x \in H^4, y \in H^0$ admitting δ close to this bound.

In fact, these upper bounds on admissible δ are actually quasi-admissible δ that can be obtained in a straightforward manner. For example, suppose we restrict to x of degree 3, y of degree 0. Then for some $A, B, C, k \in \mathbb{R}$, we have

$$x(s) = s^3 + As^2 + Bs + C$$

$$y(s) = kz(s) = s^5$$

Instead of trying to find admissible δ using this x and y, we will try to find quasi-admissible δ. That is, we want δ such that

$$z(s) = (s^2 - 2\delta s + 1)x(s) + (s^2 - 1)y(s) \in \overline{H}.$$

In other words, this $z(s)$ can be quasi-stable instead of just stable. Note that $z(s)$ must be of degree 5. We will specify a form for $z(s)$ that ensures it is quasi-stable. Consider the case $z(s) = s^5$. This is clearly quasi-stable as its only roots are at $s = 0$. To ensure that $z(s) = s^5$ and equation (3.1) holds, we require

$$(s^2 - 2\delta s + 1)(s^3 + As^2 + Bs + C) + (s^2 - 1)k = s^5$$

Equating coefficients gives us the following 5 equations in 5 unknowns.

$$A - 2\delta = 0$$

$$-2A\delta + B + 1 = 0$$

$$A - 2B\delta + C + k = 0$$

$$B - 2C\delta = 0$$

$$C - k = 0$$

In fact, ensuring that we have as many equations as unknowns was part of the motivation for letting $z(s) = s^5$. Solving for A, B, C, k, δ, we find

$$8\delta^4 - 8\delta^2 + 1 = 0$$

$$A = 2\delta$$

$$B = 4\delta^2 - 1$$

$$C = 4\delta^3 - 2\delta$$

$$k = 4\delta^3 - 2\delta$$

Taking the largest real root of $8\delta^4 - 8\delta^2 + 1$ gives $\delta = \sqrt{2+\sqrt{2}}/2$. Taking A, B, C, k as above yields polynomials x, y, z with real coefficients. One can verify that x is stable (via the Routh-Hurwitz test, for example), while y is degree 0 and therefore stable. Note that since $z(s) = s^5, z$ is only quasi-stable. Therefore, there is $x \in H^3, y \in H^0$ for which $\sqrt{2+\sqrt{2}}/2$ is quasi-admissible. This immediately gives the limiting value for $x \in H^3, y \in H^0$ discovered by Burke et al [17]. Combining this with Theorem 3.10, we have shown the following theorem.

Theorem 4.1. *For* $\deg(x) \leq 3$, $\delta = \frac{\sqrt{2+\sqrt{2}}}{2}$ *is quasi-admissible and all* $\delta < \frac{\sqrt{2+\sqrt{2}}}{2}$ *are admissible.*

Next, suppose that x has degree 4 and y has degree 0. For $A, k, \delta \in \mathbb{R}$, define

$$x(s) = (s^2 + 2\delta s + 1)(s^2 + A)$$

$$y(s) = k$$

Note that as long as $A \geq 0$, x will be quasi-stable and y will be stable for any k. As above, we want quasi-admissible δ. We let $z(s) = s^6$, so that $z(s)$ is quasi-stable. Finding A, δ, k amounts to solving

$$(s^2 - 2\delta s + 1)x(s) + (s^2 - 1)y(s) = z(s)$$

$$\Leftrightarrow (s^2 - 2\delta s + 1)(s^2 + 2\delta s + 1)(s^2 + A) + (s^2 - 1)k = s^6$$

$$\Leftrightarrow s^6 + (A - 4\delta^2 + 2)s^4 + (-4A\delta^2 + 2A + k + 1)s^2 + (A - k) = s^6$$

Note that the $(s^2 + 2\delta s + 1)$ term in x is used to ensure that the left-hand side will have zero coefficients in its odd degree terms. Since $(s^2 + 2\delta s + 1)$ is stable, it does not affect stability of x. Equating coefficients and manipulating, we get the following

equations.

$$16\delta^4 - 20\delta^2 + 5 = 0$$

$$A - 4\delta^2 + 2 = 0$$

$$k - A = 0$$

Taking the largest real root of $16\delta^4 - 20\delta^2 + 5$ gives $\delta = \sqrt{10 + 2\sqrt{5}}/4$. For this δ one can easily see that $A = 4\delta^2 - 2 \geq 0$, so x is quasi-stable, as are y and z by design. Once again, we were able to easily achieve the limiting value discovered by Burke et al. [17] discussed in Section 4.1 by searching for quasi-admissible δ. Combining this with Theorem 3.10, we obtain the following theorem.

Theorem 4.2. *For* $\deg(x) \leq 4$, $\delta = \frac{\sqrt{10+2\sqrt{5}}}{4}$ *is quasi-admissible and all* $\delta < \frac{\sqrt{10+2\sqrt{5}}}{4}$ *are admissible.*

The examples above demonstrate how, by considering quasi-stable x, y and z, we can find quasi-admissible δ that are limiting values of admissible δ. Moreover, the quasi-stable δ above were found by solving relatively simple algebraic equations instead of having to perform optimization over the space of stable x and y.

4.2 Algebraic specification

The observations in Section 3.4, the results in Section 3.6, and the examples in Section 4.1 all suggest the following approach we refer to as *algebraic specification*. This method will be used to find the largest known values of δ found for any given degree. We wish to construct quasi-stable $x(s), y(s), z(s)$ with repeated roots on the imaginary line

satisfying (3.1). For example, we may wish to find polynomials of the following form:

$$x(s) = (s^2 + 2\delta s + 1)(s^2 + A_1)^4(s^2 + A_2)^2(s^2 + A_3)^2(s^2 + A_4).$$

$$y(s) = k(s^2 + B_1)^3(s^2 + B_2)^2.$$

$$z(s) = s^{14}(s^2 + C_1)^2(s^2 + C_2)(s^2 + C_3).$$

We refer to such an arrangement of x, y, z as an *algebraic configuration*. As long as $\delta > 0$, the parameters $\{A_i\}_{i=1}^4$, $\{B_i\}_{i=1}^2$, and $\{C_i\}_{i=1}^3$ are all nonnegative, and k is real, $x(s), y(s), z(s)$ will be real, quasi-stable polynomials. We then wish to solve

$$(s^2 - 2\delta s + 1)x(s) + (s^2 - 1)y(s) = z(s). \tag{4.1}$$

Recall that the $(s^2+2\delta s+1)$ factor in $x(s)$ is present to ensure that the left-hand side has only even degree terms, as the right-hand side clearly only has even degree terms. Expanding (4.1) and equating coefficients, we get 11 equations in 11 unknowns. Using PHCPack [90] to solve these equations and selecting the solution with the largest δ such

that the $A_i, B_i, C_i \geq 0$, we get the following solution, rounded to seven decimal places:

$$\delta = 0.9808348$$

$$A_1 = 1.1856917$$

$$A_2 = 6.6228807$$

$$A_3 = 0.3090555$$

$$A_4 = 0.2292503$$

$$B_1 = 0.5430391$$

$$B_2 = 0.2458118$$

$$C_1 = 4.4038385$$

$$C_2 = 0.7163490$$

$$C_3 = 7.4637156$$

$$k = 196.1845537$$

The actual solution has $\delta = 0.980834821202\ldots$. This is the largest δ we have found to date using this method. By Theorem 3.10, we conclude the following theorem.

Theorem 4.3. *All $\delta \leq 0.9808348$ are admissible.*

In general, we can form an algebraic configuration for $x(s), y(s), z(s)$ as

$$x(s) = (s^2 + 2\delta s + 1) \prod_{i=1}^{m_1} (s^2 + A_i)^{j_i}. \tag{4.2}$$

$$y(s) = k \prod_{i=1}^{m_2} (s^2 + B_i)^{k_i}. \tag{4.3}$$

$$z(s) = s^c \prod_{i=1}^{m_3} (s^2 + C_i)^{\ell_i}. \tag{4.4}$$

For fixed degrees of x, y, note there are only finitely many such configurations. Instead of performing optimization over the non-convex feasible region of the Belgian chocolate problem, we instead tackle the combinatorial optimization problem of maximizing δ among the possible configurations.

Note that c in (4.4) is whatever exponent is needed to make $\deg(z) = \deg(x) + 2$. We want x, y, z to satisfy (3.1). Expanding and equating coefficients, we get equations in the undetermined variables above. As long as the number of unknown variables equals the number of equations, we can solve and look for real solutions with δ and all A_i, B_i, C_i nonnegative.

Not all quasi-stable polynomials can be formed via algebraic specification. In particular, algebraic specification forces all the roots of y, z and all but two of the roots of x to lie on the imaginary axis. However, more general quasi-stable x, y, z could have some roots with negative real part and some with zero real part. This makes the possible search space infinite and, as discussed in Section 4.1, empirically does not result in larger δ. Further evidence for this statement will be given in Section 4.4.

While the method of algebraic specification has demonstrable effectiveness, it becomes computationally infeasible to solve these general equations for very large n. In particular, the space of possible algebraic configurations of x, y, z grows almost exponentially with the degree of the polynomials. For large n, an exhaustive search over the space of possible configurations becomes infeasible, especially as the equations become more difficult to solve.

We will describe an algebraic configuration via the shorthand

$$[j_1, \ldots, j_{m_1}], [k_1, \ldots, k_{m_2}], [\ell_1, \ldots, \ell_{m_3}]. \tag{4.5}$$

This represents the configuration described in $(4.2), (4.3), (4.4)$ above. In particular, if the second term of (4.5) is empty then $y = k$, while if the third term of (4.5) is empty then z is a power of s. For example, the following configuration is given by $[3, 1], [2], [1]$:

$$x(s) = (s^2 + 2\delta s + 1)(s^2 + A_1)^3(s^2 + A_2)$$

$$y(s) = k(s^2 + B_1)^2$$

$$z(s) = s^{10}(s^2 + C_1)$$

A table containing the largest quasi-admissible δ we have found and their associated algebraic configuration for given degrees of x is given below. Note that for each entry of the table, given $\deg(x) = n$ and quasi-admissible δ, Theorem 3.10 implies that all $\hat{\delta} < \delta$ are admissible with x, y of degree at most n.

$\deg(x)$	Configuration	δ
4	[1],[],[]	0.9510565
6	[2],[1],[]	0.9629740
8	[3],[1],[1]	0.9702883
10	[3,1],[2],[1]	0.9744993
12	[3,2],[2,1],[1]	0.9764615
14	[3,2,1],[2,1],[2]	0.9783838
16	[3,2,1,1],[2,2],[2]	0.9794385
18	[3,2,2,1],[2,2],[2,1]	0.9802345
20	[4,2,2,1],[3,2],[2,2,1]	0.9808348

Figure 5: The largest known quasi-admissible δ for x, y, z designed algebraically, for varying degrees of x.

4.3 Approximating quasi-admissible δ by admissible δ

In this section we will prove Theorem 3.10. Our proof will be algorithmic in nature. We will describe an algorithm that, given δ that is quasi-admissible by quasi-stable polynomials x, y, will produce for any $\hat{\delta} < \delta$ stable polynomials $\hat{x}, \hat{y}$ admitting $\hat{\delta}$. Moreover, given $\deg(x) = n$, we will ensure that $\deg(\hat{x}) \leq n$.

of Theorem 3.10. Suppose that for a given δ there are $x, y, z \in \overline{H}$ with $\deg(x) \geq \deg(y)$ satisfying (3.1). Let $n = \deg(x)$. Define

$$R(s) := \frac{(s^2-1)y(s)}{z(s)}.$$

Note that for any $s \in \mathbb{C}$, $R(s) = 0$ iff $(s^2-1)y(s) = 0$, $R(s) = 1$ iff $(s^2-2\delta s+1)x(s) = 0$, and $R(s)$ is infinite iff $z(s) = 0$. Since x, y, z are quasi-stable, we know that for $\mathrm{Re}(s) > 0$, $R(s) = 1$ iff $s = \delta \pm i\sqrt{1-\delta^2}$ and $R(s) = 0$ iff $s = 1$. All other points where $R(s)$ is 0, 1, or infinite satisfy $\mathrm{Re}(s) \leq 0$. Precomposing $R(s)$ with the fractional linear transformation $f(s) = (1+s)/(1-s)$, we get the complex function

$$D(s) := R\left(\frac{1+s}{1-s}\right).$$

Note that this fractional linear transformation maps the unit disk $\{s||s| = 1\}$ to the imaginary axis $\{s|\mathrm{Re}(s) = 0\}$. Also note that $f^{-1}(1) = 0, f^{-1}(\delta \pm i\sqrt{1-\delta^2}) = \pm it$ where $t = \sqrt{1-\delta}/\sqrt{1+\delta}$. Therefore, $D(s)$ satisfies the following properties:

1. For $|s| < 1$, $D(s) = 0$ iff $s = 0$.

2. For $|s| < 1$, $D(s) = 1$ iff $s = \pm it$.

3. $|D(s)| < \infty$ for $|s| < 1$.

Note that the last holds by the quasi-stability of $z(s)$. Since $z(s) = 0$ implies $\mathrm{Re}(s) \leq 0$, $D(s) = \infty$ implies $|s| \geq 1$. In particular, the roots of x, y, z that have 0 real part now correspond to points $|s| = 1$ such that $D(s) = 1, 0, \infty$ respectively. For any $\epsilon > 0$, let

$$D_\epsilon(s) := D\left(\frac{s}{1+\epsilon}\right).$$

$D_\epsilon(s)$ then satisfies

1. For $|s| \leq 1$, $D_\epsilon(s) = 0$ iff $s = 0$.

2. For $|s| \leq 1$, $D_\epsilon(s) = 1$ iff $s = \pm i(1+\epsilon)t$.

3. $|D(s)| < \infty$ for $|s| \leq 1$.

Precomposing with the inverse fractional linear transformation $f^{-1}(s) = (s-1)/(s+1)$, we get

$$R_\epsilon(s) := D_\epsilon\left(\frac{s-1}{s+1}\right).$$

By the properties of $D_\epsilon(s)$ above, we find that $R_\epsilon(s)$ satisfies

1. For $\mathrm{Re}(s) \geq 0$, $R_\epsilon(s) = 0$ iff $s = 1$.

2. For $\mathrm{Re}(s) \geq 0$, $R_\epsilon(s) = 1$ iff $s = \delta_\epsilon \pm i\sqrt{1-\delta_\epsilon^2}$ where

$$\delta_\epsilon = \frac{1-(1+\epsilon)^2t^2}{1+(1+\epsilon^2)t^2}.$$

3. For $\mathrm{Re}(s) \geq 0$, $|R_\epsilon(s)| < \infty$.

Moreover, $R_\epsilon(s) \neq 0, 1, \infty$ for any s such that $\mathrm{Re}(s) < 0$. We can rewrite $R_\epsilon(s)$ as $R_\epsilon(s) = p(s)/q(s)$. Note that by the first property of R_ϵ, the only root of $p(s)$ in

$\{s|\mathrm{Re}(s) \geq 0\}$ is at $s = 1$. By properties of $f(s), f^{-1}(s)$, one can show that $p(-1) = 0$. This follows from the fact that $R(-1) = 0$, which implies that $\lim_{s\to\infty} D(s) = \lim_{s\to\infty} D_\epsilon(s) = 0$, and therefore $R_\epsilon(-1) = 0$. Therefore, $p(s) = (s^2 - 1)y_\epsilon(s)$ where $y_\epsilon(s)$ has no roots in $\{s|\mathrm{Re}(s) \geq 0\}$. By the second property of R_ϵ, the only roots of $q - p$ in $\{s|\mathrm{Re}(s) \geq 0\}$ are at $\pm\delta_\epsilon + i\sqrt{1 - \delta_\epsilon^2}$. Therefore, $q - p = (s^2 - 2\delta_\epsilon s + 1)x_\epsilon(s)$ where $x_\epsilon(s)$ has no roots in $\{s|\mathrm{Re}(s) \geq 0\}$. Finally, by the third property of R_ϵ we find that $z_\epsilon(s) = (s^2 - 2\delta_\epsilon s + 1)x_\epsilon(s) + (s^2 - 1)y_\epsilon(s)$ is stable. Moreover, basic properties of fractional linear transformations show that if $\deg(x) = n \geq \deg(y) = m$, then x_ϵ, y_ϵ are both of degree n. Therefore, $x_\epsilon, y_\epsilon, z_\epsilon$ are stable polynomials satisfying (3.1) for δ_ϵ. For any $\hat{\delta} < \delta$, we can take ϵ such that $\delta_\epsilon = \hat{\delta}$, proving the desired result. □

Note that if we start with δ admissible by stable x, y, z of degree at most n, then we can do the reverse of this procedure to perturb x, y, z to quasi-stable $\hat{x}, \hat{y}, \hat{z}$. By the reverse of the arguments above, $\hat{x}, \hat{y}, \hat{z}$ will be quasi-stable but at least one of these polynomials will not be stable. These polynomials will be associated to some quasi-admissible $\hat{\delta} > \delta$. This gives the proof of Theorem 3.11.

The proof above describes the following algorithm for perturbing quasi-stable x, y, z satisfying (3.1) to obtain stable $\hat{x}, \hat{y}, \hat{z}$ satisfying (3.1).

Input: Real numbers $\delta, \epsilon > 0$ and real polynomials $x, y, z \in \overline{H}$ satisfying (3.1).

Output: $\hat{\delta}$ and real polynomials $\hat{x}, \hat{y}, \hat{z} \in H$ satisfying (3.1).

1. Let $R(s) = (s^2 - 1)y(s)/z(s)$. For $\epsilon > 0$, compute
$$R_\epsilon(s) = R\left(\frac{(2+\epsilon)s + \epsilon}{\epsilon s + (2+\epsilon)}\right).$$
2. Reduce $R_\epsilon(s)$ to lowest terms. Suppose that in lowest terms $R_\epsilon(s) = p(s)/q(s)$.

3. Factor $p(s)$ as $(s^2-1)\hat{y}(s)$ and factor $q(s)-p(s)$ as $(s^2-2\hat{\delta}s+1)\hat{x}(s)$. Let $\hat{z}(s)=q(s)$.

To further illustrate the method of algebraic specification and this algorithm for perturbing to get quasi-stable polynomials, we give the following detailed example.

Example 4.4. *Say we are interested in x of degree 4. We may then give the following algebraic specification of x, y, z discussed in Section 4.1. In the shorthand of (4.5), this is the configuration* $[1], [], []$.

$$x(s) = (s^2+2\delta s+1)(s^2+A)$$

$$y(s) = k$$

$$z(s) = s^6$$

As in Section 4.1, we solve $(s^2-2\delta s+1)x(s)+(s^2-1)y(s)=z(s)$. This implies that δ, A, k satisfy $16\delta^4-20\delta^2+5=0$, $A=4\delta^2-2$, $k=4\delta^2-2$. Taking the largest root of $16\delta^4-20\delta^2+5$ gives $\delta=\sqrt{10+2\sqrt{5}}/4$, $A=k=(\sqrt{5}+1)/2$. Given numerically to six decimal places, $\delta=0.951057$. Computing $R(s)$ using exact arithmetic, we get

$$R(s) = \frac{(s^2-1)y(s)}{z(s)} = \frac{(s^2-1)(\sqrt{5}+1)}{2s^6}$$

We then use a fractional linear transformation $s \mapsto (1+s)/(1-s)$ to get:

$$\begin{aligned} D(s) &= R((1+s)/(1-s)) \\ &= \frac{2s(\sqrt{5}+1)(s-1)^4}{s^6+6s^5+15s^4+20s^3+15s^2+6s+1} \end{aligned}$$

One can verify that $D(s)$ can equal 1 on the boundary of the unit circle, so we push

these away from the boundary (with $\epsilon = 0.01$) by defining

$$\begin{aligned} D_\epsilon(s) &= D\big(\frac{s}{1+0.01}\big) \\ &= \frac{6.40805(0.99010s-1)^4 s}{0.942045s^6 + \ldots + 5.94054s} \end{aligned}$$

While we gave an approximate decimal form above for brevity, this computation can and should be done with exact arithmetic. We let $R_\epsilon(s) = f_\epsilon((s-1)/(s+1))$. Writing $R_\epsilon(s)$ as $p(s)/q(s)$ in lowest terms, we get:

$$p(s) = 64080.55401(0.990990s + 199.00990)^4(s^2-1)$$

$$q(s) = 0.62122 \times 10^{14} s^6 + \ldots + 0.94204$$

As proved above, $p(s)$ will equal $(s^2-1)\hat{y}(s)$. Dividing $p(s)$ by the s^2-1 factor, we get a polynomial $\hat{y}(s)$ such that its only root is at $s = -201$. Therefore $\hat{y}(s)$ is stable. The denominator, $\hat{z}(s)$ is easily verified to only have roots with negative part. Finally, the polynomial $q(s) - p(s)$ will equal $(s^2 - 2\hat{\delta}s + 1)\hat{x}(s)$. Finding its roots, one can show that $q(s) - p(s)$ only has roots with negative real part, except for roots at $s = 0.950097 \pm 0.311954i$. These roots are of the form $\hat{\delta} \pm \sqrt{\hat{\delta}^2 - 1}$ for $\hat{\delta} = 0.950097$. Therefore $\hat{\delta} = 0.950097$ is admissible via the stable polynomials $\hat{x}, \hat{y}, \hat{z}$. While we have decreased δ slightly, we have achieved stability in the process. By decreasing ϵ, we can get arbitrarily close to our original δ.

4.4 Optimality of algebraic specification

Not only does our method of algebraic specification find larger δ than have been found before, one can view previous approaches to the Belgian chocolate problem as approximating algebraic specification. In particular, previously discovered admissible δ can be

seen as approximating some quasi-admissible δ' that can be found via algebraic specification.

For example, in [20], Chang and Sahinidis found that $\delta = 0.9739744$ is admissible by

$$\begin{aligned}
x(s) &= s^{10} + 1.973511091362 61 s^9 \\
&\quad + 5.49402092964662 s^8 + 8.78344232801755 s^7 \\
&\quad + 11.67256448604672 s^6 + 13.95449016040116 s^5 \\
&\quad + 11.89912895529042 s^4 + 9.19112429409894 s^3 \\
&\quad + 5.75248874640322 s^2 + 2.03055901420484 s \\
&\quad + 1.03326203778346, \\
y(s) &= 0.00066128189295 s^5 + 3.611364710425 s^4 \\
&\quad + 0.03394722108511 s^3 + 3.86358782861648 s^2 \\
&\quad + 0.0178174691792 s + 1.03326203778319.
\end{aligned}$$

The roots of x, y, z were discussed in Section 3.4. As previously noted, x, y, z are close to polynomials with repeated roots on the imaginary axis. Examining the roots of x, y, z, one can see that x, y, z are tending towards quasi-stable polynomials x', y', z' that have the same root structure as the algebraic configuration $[3, 1], [2], [1]$. In other words, we will consider the following quasi-stable polynomials:

$$x'(s) = (s^2 + 2\delta' s + 1)(s^2 + A_1)^3(s^2 + A_2)$$

$$y'(s) = k(s^2 + B)^2$$

$$z'(s) = s^{10}(s^2 + C)$$

Solving for the free parameters and finding the largest real δ' such that $A_1, A_2, B, C \geq 0$, we obtain the following values, given to seven decimal places.

$$\delta' = 0.9744993$$

$$A_1 = 1.3010813$$

$$A_2 = 0.4475424$$

$$B = 0.5345301$$

$$C = 2.5521908$$

$$k = 3.4498736.$$

One can easily verify that taking these values of the parameters, the roots of x, y, z are close to the roots of x', y', z'. These algebraically designed x', y', z' possess the root structure that x, y, z are tending towards. Moreover, the x', y', z' show that δ' is quasi-stable and their associated δ' gives an upper bound for the δ found by Chang and Sahinidis. This demonstrates that the stable polynomials found by Chang and Sahinidis are tending towards the quasi-stable ones listed above. Moreover, by Theorem 3.10 all $\delta < 0.9744993$ are admissible.

In fact, many examples of admissible δ given in previous work are approximating quasi-admissible δ found via algebraic specification. This includes the previously mentioned examples in [17] and all admissible values of δ given by Chang and Sahinidis in

[20]. We further conjecture that for all admissible δ, there is a quasi-admissible $\delta' > \delta$ that can be achieved by algebraically specified x, y, z.

More formally, if we fix x, y to be of degree at most n, let δ_n^* denote the supremum of the optimization problem in (3.2). Note that as discussed in Section 3.6, δ_n^* is not admissible by x, y of degree at most n. The empirical evidence given in this section and in Sections 3.4 and 4.1 suggests that this δ_n^* is quasi-admissible and can be obtained through algebraic specification. This leads to the following conjecture.

Conjecture 4.5. *For all n, δ_n^* is quasi-admissible by some x, y, z that are formed via algebraic specification.*

Recipe 7: Chocolate Raspberry Cake

Cake Ingredients

- 3 ounces bittersweet chocolate
- 1 $\frac{1}{2}$ cups hot coffee
- 3 $\frac{1}{2}$ cups sugar
- 2 $\frac{1}{2}$ cups flour
- 1 $\frac{1}{2}$ cups cocoa powder
- 2 teaspoons baking soda
- $\frac{3}{4}$ teaspoon baking powder
- 1 $\frac{1}{4}$ teaspoons salt
- 3 eggs
- $\frac{3}{4}$ cup vegetable oil
- 1 $\frac{1}{2}$ cups buttermilk
- $\frac{3}{4}$ vanilla
- 20 ounces frozen raspberries
- 2 tablespoon cornstarch
- Chocolate ganache or frosting

Preparation

1. Preheat the oven to 300° F. Line bottom of two round cake pans with

parchment paper and grease.

2. Finely chop chocolate and combine with hot coffee. Stir occasionally until chocolate is melted and smooth.

3. In a large bowl, sift together 3 cups sugar, flour, cocoa powder, baking soda, baking powder, and salt.

4. In another bowl, mix egg yolks until thickened slightly. Slowly add oil, buttermilk, vanilla, and chocolate mixture in stages, while mixing. Combine well. Add sugar mixture and mix until just combined.

5. Divide batter evenly between pans and bake for 1 hour or until a tester inserted into the center comes out clean. Cool completely in pans before removing.

6. Puree raspberries in a blender or food processor. Press through a fine mesh with the back of a spoon to remove seeds. Add remaining sugar and cornstarch and heat in a small pot until the mixture boils. Stir constantly. Let cool and spread on top ofone of the two cake layers.

7. Place second cake on top of the other and frost to taste.

Part II

Subspace Clustering

Chapter 5

The Subspace Clustering Problem

Recipe 8: Croque-Madame

Ingredients

- 5 tablespoons butter
- 1 tablespoon flour
- $\frac{2}{3}$ cup milk
- Salt
- Nutmeg
- 4 thick slices of country bread
- 4 slices ham
- 2 slices Gruyère cheese
- 2 eggs

Preparation

1. Preheat oven to 300° F and preheat a cast-iron skillet on the stove.
2. In a small saucepan, over medium heat, melt 1 tablespoon butter. When bubbles have subsided, add flour and whisk for 1 minute. Slowly whisk in milk and bring to a boil. Remove from heat and season with salt and nutmeg.
3. Spread two slices of bread with sauce. Lay ham on top of each and top with cheese. Top each with a slice of bread.
4. Melt remaining butter and brush both sides of sandwiches with butter. Place sandwiches cheese side down in skillet and cook until brown. Flip and repeat. Transfer skillet to oven and bake until cheese is bubbling. Remove sandwiches from skillet.

5. Fry eggs with a small amount of butter. Slide one fried egg on top of each sandwich. Serve hot.

5.1 Background

5.1.1 Clustering

Discussions of machine learning, especially news articles touting the success of machine learning methods, often focus only on *supervised learning*. In supervised learning, we have a data set with *labels*, and we wish to predict the labels from the data. For example, suppose we have a collection of images featuring either a cat or a dog. The data are the images themselves, while the labels are cat or dog depending on the image. We would like to train a machine learning algorithm to be able to look at a new, unlabeled image of a cat or dog and correctly determine which animal is featured in the image. We do so by training the algorithm on enough labeled data, in the hopes that the algorithm can eventually recognize features that images of cats may possess (such as pointy ears and triangular noses) and that images of dogs may possess (such as floppy ears and longer snouts). An example of such a labeled data set is given in Figure 6 below.

Unfortunately, many datasets have few or no labels. This can be because the labels are too expensive to acquire (such as in professional medical diagnosis) or because they are simply too difficult to acquire in a reasonable span of time. In such cases, we may wish to use machine learning to understand our data better, even when we do not have labels that we wish to predict. This is referred to broadly as *unsupervised learning*. In

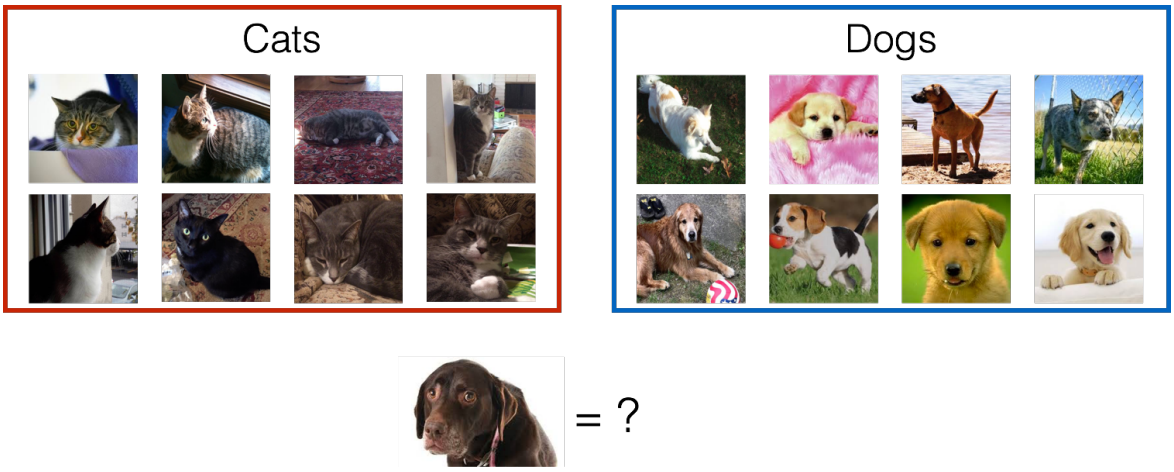


Figure 6: A labeled dataset of cats (upper-left) and dogs (upper-right). In a supervised learning task, we may wish to predict the label of the image below.

unsupervised learning, the goal is to design algorithms that can infer patterns and useful information about our data, without access to any labels of our data. Suppose again that we had images of cats and dogs, but this time without any knowledge of what animal is featured in each image. To better understand our data, we might want to group the images according to similarity. We can do this without knowing the individual sentence structure of each image by instead looking at similarities between the images. We might hope that even without direct knowledge of which images are cats and dogs, the algorithm can implicitly group the images into cats and dogs, or that it groups them by breed. If the algorithm can correctly do this, then it becomes much easier to understand our data by simply looking at a representative image from each group. An example is given in Figure 7 below.

Grouping unlabeled data by some notion of similarity is referred to as *clustering*, and has been of great interest to machine learning theorists and data scientists over the last two decades. This has sparked the development of a wide variety of clustering algorithms, all with different measures of success for different scenarios. Two of the

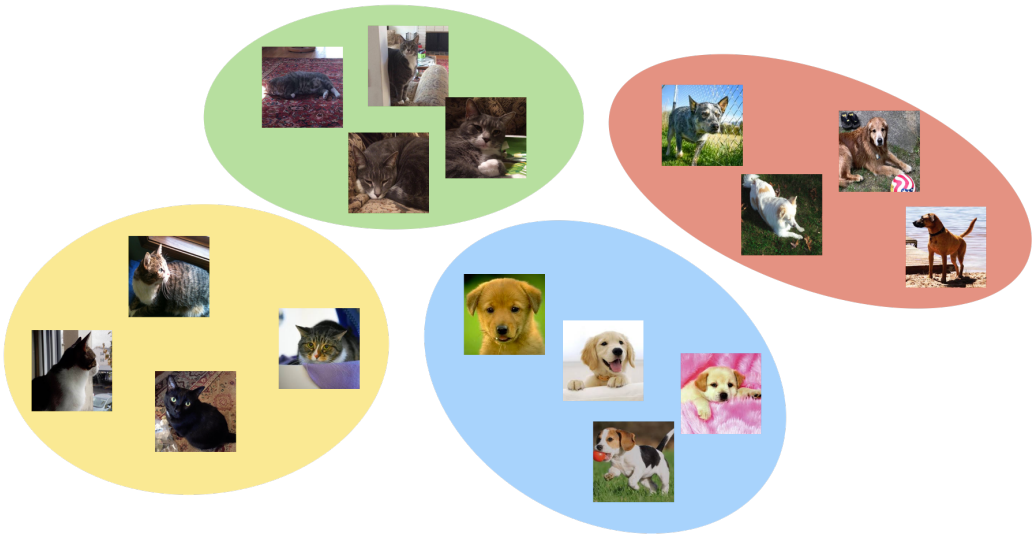


Figure 7: An unlabeled dataset of cats and dogs plotted in 2 dimensions. We also give hypothetical clusters of this data set, denoted by various colors. Note that from these colors, we can see more easily which are cats and which are dogs.

most prominent such methods are *k-means clustering* and *spectral clustering*. To make these methods work, however, we need some notion of distance between our data points, where the distance accurately captures how dissimilar any two data points are.

In many practical applications, we would like to cluster our points according to a certain idea, but may not have an obvious choice of distance between the data points. As a motivating example, imagine we have a set of 2-dimensional points. Moreover, suppose that while the points do not have any explicit structure, they lie on the union of a small number number of lines through the origin.

If we only view the data points, then we do not immediately know what lines the points lie on. We would like to group the points according to which line they lie on and recover the underlying lines. In this case, standard clustering algorithms may fail us because they tend to prioritize fat, ball-like shapes instead of thin linear shapes. Figure

8 below illustrates why standard clustering algorithms often fail in this setting.

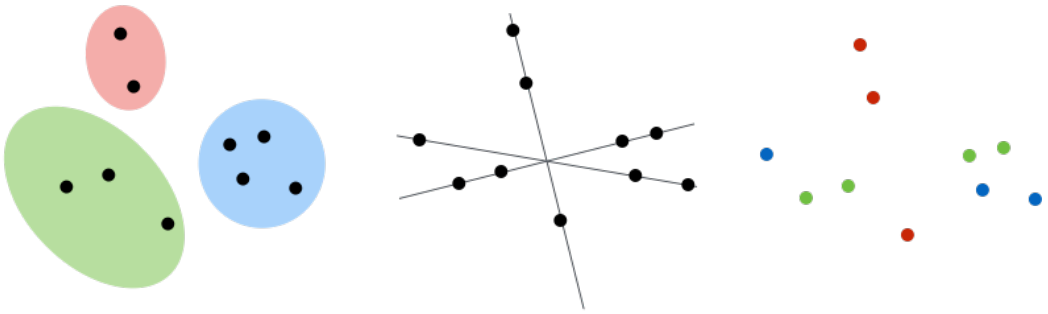


Figure 8: An example of clustering points lying on 2-dimensional lines. On the left, we show the groupings obtained by standard clustering algorithms. In the middle, we show the underlying lines, and on the right we show the true clusters we wish to find.

This kind of task is a special case of *subspace clustering*. In general, we may have points lying on larger-dimensional shapes, such as planes, and wish to recover the underlying shapes and group the points accordingly.

5.1.2 Matrix Completion

To make matters worse, in practical applications we might have missing data. If you imagine each data point as a list of values, some of these values may be altered by noise (*i.e.,* changed in some way artificially), or else missing entirely. When conducting a survey, for example, many respondents do not answer all the questions (in which case we have missing data), or fill out a question inaccurately (in which case we have noisy answers). Subspace clustering when there is missing data or noise can be thought of as a generalization of a famous problem in machine learning, *matrix completion*.

To understand this problem, we will imagine that we are Netflix, a popular movie streaming service. We have access to an enormous amount of customer data that tells us whether a given user rated a given movie highly. We compile all this information into a

matrix, a rectangular array of values. In this case, imagine that the columns correspond to users, while the rows correspond to movies. The entry in a given row and column is a 1 if the user liked the movie, and a 0 if they did not like the movie. The result might look something like Figure 9 below.

$$\begin{array}{c} \\ \text{Movies} \end{array} \begin{array}{c} \text{Users} \\ \begin{bmatrix} 1 & ? & 0 & 1 & ? & 1 \\ 0 & ? & 1 & 1 & 0 & ? \\ ? & 0 & ? & 0 & ? & 1 \\ 1 & 1 & 0 & ? & 1 & ? \end{bmatrix} \end{array}$$

Figure 9: A matrix representing hypothetical user-movie preferences. A 1 indicates the user rated a given movie positively, a 0 indicates the user rated a given movie negatively, and a ? means that the user has not rated that movie

Since not every user has seen and rated every movie, much of this matrix is missing. It is then our job to predict the missing entries of the matrix. This allows us to recommend movies to users that we think they would enjoy. How to fill in the missing values in a way that makes sense is referred to as the *matrix completion* problem.

If we make no assumptions whatsoever on the structure of the matrix, then the missing data could take any value. Instead, we often assume that the matrix is *low-rank*. This is a mathematical property that roughly translates to, in the Netflix problem, that there are a small number of archetypal users and that every other user's movie preferences can be represented as a combination of the preferences of the archetypal users. For example, there may be the "comedy user", the "romance user", the "horror user", and the "action user", who all prefer their eponymous genres. The low rank

assumption says that everyone's preferences can be represented in terms of these four users' preferences. A given user may be one-half comedy, one-fourth romance, one-fourth action, and zero horror.

As it turns out, this assumption allows us to efficiently and accurately perform matrix completion. The subspace clustering relaxes this assumption. It instead says that there are multiple sets of archetypal users, and that every user can be represented as a combination of the archetypal users from one of these sets. If there are enough sets, standard matrix completion techniques fail. We therefore need to derive new and efficient methods to tackle this setting.

5.2 Prior Work

In many applications, including image compression [44, 101], network estimation [35], video segmentation [22, 47], and recommender systems [104], what is ostensibly high-dimensional data can be modeled as data sampled from a *union of low-dimensional subspaces*. In subspace clustering, we observe a data matrix $X \in \mathbb{R}^{n \times N}$ whose columns lie near a union of several low-dimensional subspaces of $\mathbb{R}^n$. We wish to cluster the columns according to their subspace and infer the subspaces. In practice, X may be corrupted by large amounts of noise, adversarial or otherwise, or it may have missing entries.

Subspace clustering has experienced significant attention over the last decade, resulting in many different algorithms with varying levels of established theory. These include expectation-maximization methods [14], algebraic methods [92], matrix factorization methods [22], and local sampling methods [77], among others. Of particular

note is sparse subspace clustering (SSC) [31], which exhibits good empirical performance in many real data applications and enjoys provable guarantees on its performance [86]. Moreover, various theoretical and empirical work show that SSC can handle high-dimensional data matrices [86], outliers [87], and some forms of noisy measurements [96].

When there are missing entries, subspace clustering is a generalization of low-rank matrix completion [19]. Unlike low-rank matrix completion, the data matrix may have high rank if there are many subspaces. While there are many algorithms for low-rank matrix completion with theoretical guarantees on convergence and correctness [19], such analysis has been more elusive for subspace clustering. Various algorithms for subspace clustering with missing data have been proposed [40, 91, 102, 74, 30]. Many of these exhibit strong empirical results but lack theoretical guarantees. On the other hand, [34] gives a theoretically justified method for subspace clustering with missing data, but require an unrealistically large number of samples. [75] gives information-theoretic lower bounds on the number of observations per column required, but it is not known whether the aforementioned methods meet these bounds.

Methods for subspace clustering with missing data typically require knowledge of the location of the missing entries. Works like [30, 102] even exploit such knowledge in the design of their estimators. In practice, such information may not be available. In some applications, we face *presence-only* data, where we only record the observed presence of a feature [73]. In ecological modeling, we often only have access to the observed population presence of a species in a given location but we do not know when a species is absent [97]. In [18], the authors acknowledge that in structured mammography data, there is ambiguity in whether to interpret zeros as missing or as indicating that a breast

imaging radiology feature is actually not present. This ambiguity is common in many models and applications [59, 33, 36], yet many proposed subspace clustering algorithms cannot be used in this setting.

5.2.1 Our Contributions

In this work, we present and analyze a generalization of SSC, LS-SSC. The algorithm is explained below in full, but involves using a Frobenius-norm loss functions, in addition to the standard SSC regularization term. We provide analysis for LS-SSC and give theoretical guarantees on its success. We give a deterministic criteria for success based on the geometry of the true samples and the noise level defined as the maximum ℓ_2 norm of the additive error in each observation. The guarantee does not require any distributional or structural assumptions on the additive noise. This allows us to show that LS-SSC succeeds in the presence of missing data under the random subspaces, random samples model. In such settings, we give an explicit bound on the number of missing entries per column that the algorithm can tolerate.

One important facet of our work is that it is *location-agnostic*. The algorithm does not require explicit knowledge of the location o the missing data. This allows these estimators to be used in more general situations than algorithms such as those in [30, 102] that require the locations of the missing entries. Moreover, our theoretical guarantees all still work when we do not know the locations of the missing data.

5.3 Problem Statement

We are given a matrix $X \in \mathbb{R}^{n \times N}$, where X is the sum of an uncorrupted data matrix Y and a noise matrix Z. We can view subspace clustering with missing data as a special case of this setup where the unobserved entries of Y are replaced by 0 to obtain X. This is equivalent to the setting where the noise matrix Z satisfies $Z_{ij} = -Y_{ij}$ for each missing entry (i, j). We do not assume that we know the locations of the missing data. This way, we allow for a zero entry in X to correspond to an actual zero or to a missing entry. We assume that the columns of Y come from a union of L low-dimensional subspaces

$$S_1 \cup S_2 \cup \ldots \cup S_L.$$

We make no assumptions on how the subspaces are aligned so that they can intersect arbitrarily. We do not assume that we know the underlying value of L.

Goal: We wish to find clusters $C_1, \ldots, C_L$ where C_ℓ consists of the indices of all columns of Y drawn from S_ℓ. In other words, we wish to cluster our observations according to their true underlying subspace.

As discussed above, a great deal of work has been devoted to subspace clustering, especially when the observations are uncorrupted by noise or missing data. Below we present one of the most successful approaches, sparse subspace clustering (SSC), first proposed in [31].

5.3.1 Sparse Subspace Clustering

Suppose that $X \in \mathbb{R}^{n \times N}$ is uncorrupted by noise. Therefore, the columns of X lie in a union of low-dimensional subspaces $S_1 \cup S_2 \cup \ldots \cup S_L$. Consider any column x drawn

from S_1. The key insight in [31] is that X is *self-expressive*. To understand this, consider the case that X does not come from a union of low-dimension subspaces, but is instead a full rank $n \times N$ matrix. Then, to represent any column x as a linear combination of other columns in X, we would typically need to use n distinct columns. However, if x comes from S_1 and enough observations from S_1 are present in X, then we can represent x as a linear combination of at most $\dim(S_1)$ other columns. If $\dim(S_1)$ is much smaller than n, then this amounts to a *sparse* representation of x in terms of other columns.

For simplicity, suppose all the S_ℓ have dimension d. Then using the reasoning above, if we have enough observations from each subspace then we can represent each column x as a sparse linear combination of d other columns that are drawn from the same subspace. Let $c \in \mathbb{R}^N$ represent the coefficients of this sparse linear combination. Then c generally encodes other columns in X that come from the same subspace as x. If we amalgamate all the c into a matrix C, then C will approximately have a block structure, where each block corresponds to the columns drawn from a given subspace. In particular, $W = |C| + |C|^T$ will be an *affinity matrix* with a block structure. Here, W being an affinity matrix simply means that W_{ij} represents how the affinity between x_i, x_j. High affinity means that there is a greater chance that they are drawn from the same subspace. We can then use standard clustering algorithms on this matrix to recover the subspace clusters.

Suppose we wish to find the sparse linear combination c_i associated to x_i. We could solve the following optimization problem:

$$\min_c \|c\|_0 \ \text{ s.t. } \ x_i = Xc_i, \ c_i = 0.$$

Note that here $\|c\|_0$ denotes the number of non-zero elements in c. We enforce $c_i = 0$,

as otherwise we could simply let c_i be the vector with zeros everywhere and a 1 in entry i. While the solution to this problem would give us a sparse linear combination, this problem is not efficiently solvable. In particular, $\|\cdot\|_0$ is a non-convex, non-continuous function. As a result, we wish to use a *convex relaxation* of the function. We typically use the ℓ_1 norm as a convex relation of $\|\cdot\|_0$, as it is known to often find sparse solutions to undetermined linear systems of equations [26]. Therefore, we instead solve, for every column i,

$$\min_c \|c\|_1 \quad \text{s.t.} \quad x_i = Xc_i,\ c_i = 0.$$

We can perform all N of these optimization problems simultaneously by solving the following:

$$\min_C \|C\|_{1,1} \quad \text{s.t.} \quad X = XC,\ \text{diag}(C) = 0. \tag{5.1}$$

SSC then uses spectral clustering [70] on the affinity matrix $W = |C| + |C|^T$. The clusters of W correspond to subspace clusters of the original matrix X. If we approximately recover the sparse representations discussed above, then spectral clustering on the graph with edge weights W will recover the correct clusters with high probability [95].

5.3.2 LS-SSC

While there are a great deal of empirical and theoretical guarantees for SSC when X is uncorrupted by noise, we have to change our approach when X has missing data or noise added. In this setting, the X is no longer self-expressive. For example, consider a column x. Note that $x = y + z$, where y is the true observation and z is some noise. Since the matrix Y is self-expressive, we know that there is a sparse linear combination

c such that $y = Yc$. Therefore,

$$\begin{aligned} x - Xc &= (y + z) - (Y + Z)c \\ &= y - Yc + z - Zc \\ &= z - Zc. \end{aligned}$$

Since c is sparse, if each column of Z is not too large and each entry of c is not too large, then $x - Xc$ will still be relatively small. In particular, we could try to minimize $\|x - Xc\|$ according to some norm in addition to enforcing the self-expressive property of c. That is, we could solve, for each column x_i, the following optimization problem:

$$\min_c \|c\|_1 + \lambda \|x - Xc\|_2^2 \text{ s.t. } c_i = 0. \tag{5.2}$$

Here λ is a parameter that can be tuned according to the problem. Note that the fact that we use $\|c\|_1$ instead of $\|c\|_0$ helps not just to make this more efficient, but it also helps limit the size of the entries of c, which in turn allows for $x - Xc$ to be smaller.

Amalgamating (5.2) into a single optimization problem, we get the following problem:

$$\min_C \|C\|_{1,1} + \frac{\lambda}{2} \|XC - X\|_F^2 \text{ s.t. } \operatorname{diag}(C) = 0. \tag{5.3}$$

This optimization will be part of the full algorithm LS-SSC. Here LS stands for least-squares, since the optimization above involves a least-squares loss. This variant has been previously considered in the literature with different theoretical guarantees [87, 96] or with only empirical evidence for its effectiveness [102].

This gives us the optimization problems for LS-SSC. We then apply spectral clustering to the weighted graph G with affinity matrix $W = |C| + |C|^T$. The resulting clusters are the subspace clusters we return. We use the standard technique of estimating the number of clusters $\hat{L}$ from the spectrum of the normalized Laplacian associated

to W [93]. The full algorithm for LS-SSC is given below.

Algorithm 4: LS-SSC

Input: A data matrix $X \in \mathbb{R}^{n \times N}$ and $\lambda > 0$.

1. Solve
$$\min_{C} \ \|C\|_{1,1} + \lambda \|X - XC\|_F^2 \ \text{ s.t. } \ \text{diag}(C) = 0$$
2. Form the weighted graph G on N vertices with affinity matrix $W = |C| + |C|^T$.
3. Let $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_N$ denote the eigenvalues of the normalized Laplacian of G. Set
$$\hat{L} = N - \underset{i=1,\ldots,N-1}{\text{argmax}} (\sigma_i - \sigma_{i+1}).$$
4. Apply spectral clustering to G with $\hat{L}$ clusters.

Output: Clusters $\mathcal{X}_1, \ldots, \mathcal{X}_{\hat{L}}$.

There are many good existing ways of solving LS-SSC computationally. [96] describes a method to solve (5.3) using a modification of the ADMM method [13]. Moreover, [87] shows that TFOCS [7] has competitive performance in solving this optimization program in practical applications.

One important facet of the optimization program above is that under many common missing data models, we do not need to know the locations of the missing entries. When data in X is missing, we often assume that it is replaced by a zero or some other value. Note that if we do know the missing entry locations then we can always construct this *entry-wise zero-fill*. In the event that we have such a matrix, we do not explicitly need to encode where these zeros are. As discussed above in, this allows us to use these algorithms to handle *presence-only data*, where the presence of a zero could indicate a lack of a certain feature, or it could indicate that the presence of this feature was not tested [73, 59, 33, 36].

Let C denote the output of (5.3). Then C_{ij} ideally encodes whether columns i and j come from the same subspace. In order for C to be a good estimate, we want it to

have no *false positives*, that is, we want $C_{ij} = 0$ if i and j do not correspond to the same subspace. Before elaborating on related works, we state informal versions of our main results in both deterministic and random settings.

We wish to find conditions for which LS-SSC succeeds with high probability. A good proxy for this is to show that C has no false positives [93], that is, for any i and j corresponding to different subspaces, $C_{ij} = 0$. This is reflected in the following definition.

Definition 5.1 ([86])**.** *We say that X obeys the subspace detection property with parameter λ if for all ℓ and for all x_i drawn from S_ℓ, the columns c_i of the solution to (5.3) has non-zero entries corresponding only to columns in Y sampled from S_ℓ. We say that the subspace detection property holds if there is a non-empty interval of values of λ for which the subspace detection property with parameter λ holds.*

The above definition states that c_i does not contain any entries corresponding to subspaces that x_i was not drawn from. If the c_i are non-zero and the subspace detection property holds, we should achieve low clustering error. In practice, as long as the false positive entries of c_i much smaller than the true positives, and we have enough true positives, we will achieve low clustering error. We will find conditions for which the subspace detection property holds and the c_i are non-trivial.

5.4 Mathematical Perspectives and Summary of Results

We somehow wish to show that when solving (5.3), we get a matrix C with the subspace detection property. As it turns out, we will be able to use *duality* in optimization to do so. Important duality techniques will allow us to show that if there are *dual vectors* satisfying certain geometric conditions, then the subspace detection property will hold. Ensuring that these dual vectors satisfy the desired geometric properties hold will constitute the bulk of our work.

These dual vectors will be defined in terms of *dual norms* and the dual optimization problem (5.3), while the geometric conditions will involve bounding the inner product of corrupted observations x and these dual vectors. We also require the use of notions such as *subspace incoherence*. This is similar, but notably different, to the notion of subspace incoherence in [86]. This concept will measure how aligned a given subspace is with the corrupted observations drawn from other subspaces. We will also require from convex analysis, such as the inradius of a convex body. This will be used to control geometric properties of the dual direction. The convex analysis comes about since both the primal and dual problem in LS-SSC are convex, and therefore have important underlying geometric properties we can analyze. These concepts all control the geometry underlying the true observations y, the noise z, and their relation to the dual directions. This allows us to derive the following result concerning LS-SSC. This is an informal version of Theorem 6.6 below.

Theorem 5.2. *Suppose we are given $X = Y + Z$ where Y comes from a union of subspaces model and each column of Z has ℓ_2 norm bounded by an explicit function of*

the configuration of subspaces and the placement of true samples on subspaces. Then there is an explicit interval of λ for which LS-SSC returns no false positives.

When the y are drawn by some random process, we would like to derive high-probability results on whether LS-SSC satisfies the theorem above. To do so, we use theory from high-dimensional statistics and geometric functional analysis. We show in Theorem 6.7 below that if the subspaces are drawn according to the uniform measure on all d-dimensional subspaces (in other words, according to the standard measure on the associated Grassmannian) when d is not too large, and the noise obeys a geometric bound on its size, then LS-SSC will succeed with high probability.

To further analyze the setting that we have missing data, we can draw connections between the entry-wise zero fill of a vector with missing data to taking a random projection of a vector on to a lower-dimensional axis-aligned subspace. We can then apply Theorem 6.7 and results about the behavior of random vectors under such projections to derive the following theorem. This is an informal version of Theorem 6.8 below.

Theorem 5.3. *Assume that Y is an $n \times N$ matrix whose columns are generated uniformly at random from a number of d-dimensional subspaces chosen uniformly at random. Suppose we are given an incomplete version X of Y where zeros have been filled into the missing locations. If $d \leq \tilde{O}(n/\log N)$ and each column is missing at most $\tilde{O}(n/d)$ entries, then there is an explicit interval of λ for which LS-SSC returns no false positives with high probability.*

Here $\tilde{O}$ hides a logarithmic dependence on the number of samples drawn from each subspace.

Chapter 6

Subspace Clustering with Missing and Corrupted Data

Recipe 9: Gougères

Ingredients

- $\frac{1}{2}$ cup whole milk
- $\frac{1}{2}$ cup water
- 1 stick butter
- $\frac{1}{2}$ teaspoon salt
- 1 cup flour
- 5 eggs
- 1 $\frac{1}{2}$ cups coarsely grated Gruyère cheese

Preparation

1. Preheat oven to 425° F. Line two baking racks with parchment paper.
2. Bring milk, water, butter, and salt to a rapid boil. Add flour and lower heat to medium-low. Whisk until a dough comes together.
3. Turn dough out into bowl. Add eggs one at a time and beat until the dough is thick and shiny. Beat in grated cheese.
4. Spoon 1 tablespoon of dough on to the baking racks for each gougère. Put racks in oven and turn temperature down to 375° F. Bake for 12 minutes, rotate pans, then bake for another 12 minutes.

6.1 Preliminaries

6.1.1 Dual Programs and Convex Geometry

Let $Y^{(\ell)} \in \mathbb{R}^{n \times N_\ell}$ denote the submatrix of columns drawn from S_ℓ. We let $X^{(\ell)}$ and $Z^{(\ell)}$ denote the corresponding column submatrices of X and Z w. Let d_ℓ be the dimension of S_ℓ and let N_ℓ denote the number of columns in Y drawn from S_ℓ. We define $\kappa_\ell = N_\ell / d_\ell$.

For a matrix $A \in \mathbb{R}^{n \times m}$, we let A_{-i} denote the $n \times (m-1)$ submatrix formed by removing the ith column. We let $\mathcal{Y} \subseteq \mathbb{R}^n$ denote the set of columns of Y and let $\mathcal{Y}^{(\ell)}$ be the set of columns in X corresponding to $S^{(\ell)}$. We define $\mathcal{X}$ and $\mathcal{X}^{(\ell)}$ analogously.

For a matrix A, let $\mathcal{SC}(A)$ denote the symmetrized convex hull of its columns. If A has columns $a_1, \ldots, a_n$, then $\mathcal{SC}(A)$ is $\text{conv}(\pm a_1, \ldots, \pm a_n)$. We write $\mathcal{Q}_{-i}^{(\ell)}$ to denote $\mathcal{SC}(Y_{-i}^{(\ell)})$. Finally, we require some definitions from convex analysis.

Definition 6.1. *Given a set $\mathcal{P} \subseteq \mathbb{R}^d$, the polar set $\mathcal{P}^\circ$ of $\mathcal{P}$ is defined as*

$$\mathcal{P}^\circ = \{y \in \mathbb{R}^d | \langle x, y \rangle \leq 1 \text{ for all } x \in \mathcal{P}\}.$$

Note that $\mathcal{P}^\circ$ is a convex region.

Definition 6.2. *For any closed polytope $\mathcal{P}$, we let $r(\mathcal{P})$ denote the inradius of $\mathcal{P}$. This is defined as the radius of the largest Euclidean ball that can be inscribed in $\mathcal{P}$.*

Definition 6.3. *For any closed polytope $\mathcal{P}$ and subspace S, we let $r_S(\mathcal{P})$ denote the relative inradius of $\mathcal{P}$ with respect to S. This is defined as the radius of the largest disk in S that can be inscribed in $\mathcal{P}$.*

Suppose $\mathcal{P} \subseteq \mathbb{R}^n$ is symmetric and convex. Here symmetric means that $\mathcal{P} = -\mathcal{P}$. Note that for such polytopes, the largest inscribed ball will necessarily be centered at 0. This follows from the fact that if a ball B of radius r can be inscribed into $\mathcal{P}$, then by symmetry, so can $-B$. Taking the convex hull of $B \cup -B$, we necessarily contain the ball of radius r centered at 0.

Let B_r denote the Euclidean ball centered at 0 of radius r in $\mathbb{R}^n$. Then for $\mathcal{P}$ symmetric and convex, we have

$$r(\mathcal{P}) = \sup\{r : B_r \subseteq \mathcal{P}\}.$$

$$r_S(\mathcal{P}) = \sup\{r : B_r \cap S \subseteq \mathcal{P}\}.$$

We now define the circumradius of such $\mathcal{P}$.

Definition 6.4. *For any closed polytope $\mathcal{P}$, the relative circumradius of $\mathcal{P}$, denoted $R(\mathcal{P})$ is the radius of the Euclidean ball containing $\mathcal{P}$.*

Definition 6.5. *For any closed polytope $\mathcal{P}$, the relative circumradius of $\mathcal{P}$, denoted $R(\mathcal{P})$ is the radius of the smallest disk in S containing $\mathcal{P}$.*

Using the same notation as above, and again assuming $\mathcal{P}$ is symmetric and convex, we have

$$R(\mathcal{P}) = \inf\{r : B_r \supseteq \mathcal{P}\}.$$

$$R_S(\mathcal{P}) = \inf\{r : B_r \cap S \supseteq \mathcal{P}\}.$$

For notational convenience, we will define

$$r_\ell := \min_{i:x_i \in \mathcal{X}_\ell} r_{S_\ell}(\mathcal{Q}_{-i}^{(\ell)}).$$

$$R_\ell := \max_{i:x_i \in \mathcal{X}_\ell} R_{S_\ell}(\mathcal{Q}_{-i}^{(\ell)}).$$

6.1.2 Dual Directions and Incoherence

Given a vector x and a matrix A, we define an optimization problem denoted $P(x, A, \lambda)$ as

$$\min_{c,e} \|c\|_1 + \frac{\lambda}{2}\|e\|_2^2 \;\text{ s.t. }\; e = x - Ac, \tag{6.1}$$

and its Lagrangian dual $D(x, A, \lambda)$ is given by

$$\max_{\nu} \langle x, \nu \rangle - \frac{1}{2\lambda}\|\nu\|_2^2 \;\text{ s.t. }\; \|A^T\nu\|_\infty \leq 1. \tag{6.2}$$

The optimization problem for LS-SSC in (5.3) is equivalent to solving $P(x_i, X_{-i}, \lambda)$ for $i = 1, \ldots, N$. We will use D to analyze the geometry underlying our problem. For a given x, A, λ, let ν be the solution to $D(x, A, \lambda)$. If there are multiple solutions, select the one with the smallest ℓ_2 norm. The corresponding dual *direction* v is defined by

$$v(x, A, \lambda) = \nu / \|\nu\|_2.$$

Define

$$v_i^{(\ell)} \coloneqq v(x_i^{(\ell)}, X_{-i}^{(\ell)}, \lambda) \;\text{ and }\; V^{(\ell)} \coloneqq [v_1^{(\ell)}, \ldots, v_{N_\ell}^{(\ell)}].$$

We say that the set $\mathcal{X}_\ell$ is μ-*incoherent* with respect to the set $\mathcal{X} \backslash \mathcal{X}_{-i}^{(\ell)}$ if

$$\mu \geq \mu(\mathcal{X}_\ell) \coloneqq \max_{y \in \mathcal{Y} \backslash \mathcal{Y}^{(\ell)}} \|(V^{(\ell)})^T y\|_\infty. \tag{6.3}$$

For notational convenience, we will often write μ_ℓ for $\mu(\mathcal{X}_\ell)$. This parameter μ is referred to as the *subspace incoherence*. It is a measure of the alignment between the true observations $\mathcal{Y}^{(i)}$ from each subspace S_i and the corrupted observations $\mathcal{X}^{(j)}$ from S_j for $j \neq i$. Intuitively, the smaller μ is, the less aligned $\mathcal{Y}^{(i)}$ and $\mathcal{X}^{(j)}$ are. If μ is small enough, then it should be easier to group observations from distinct subspaces into distinct clusters. Below, we give a pictorial explanation of the subspace incoherence.

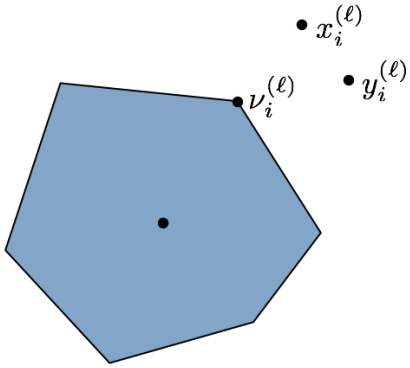


Figure 10: The dual direction $v(x_i^{(\ell)}, X_{-i}^{(\ell)}, \lambda)$, where $x_i^{(\ell)}$ is a corrupted version of the true observation $y_i^{(\ell)}$.

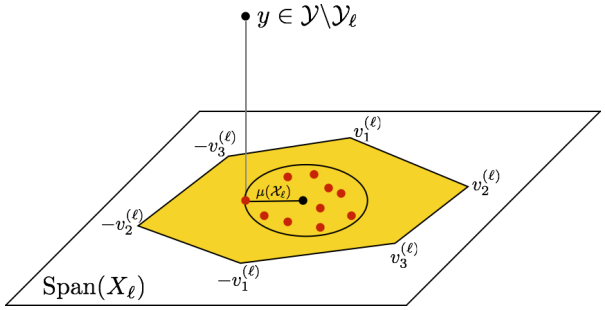


Figure 11: The subspace incoherence μ_ℓ is the radius of the smallest sphere in the span of $X^{(\ell)}$ containing all projections of $y \in \mathcal{Y} \backslash \mathcal{Y}^{(\ell)}$ onto the polytope determined by the dual directions.

This definition of subspace incoherence is a generalization of the subspace incoherence defined by [86] to the noisy setup described. If there is no noise in the samples then for λ sufficiently large these definitions of subspace incoherence will specialize to the definition in [86].

6.2 Main Results

6.2.1 Deterministic Model

Consider a matrix of samples Y coming from some fixed subspaces, and let Z be a deterministic noise matrix. We observe $X = Y + Z$. We assume that each column of X has at least 1 non-zero entry. Define

$$\delta = \max_i \|z_i\|_2.$$

We will characterize how large δ can be for the subspace detection property to hold. For ease of analysis we assume, as in [86], that each column y of Y lies on the unit sphere.[1]

The following theorem gives conditions on the subspaces and noise under LS-SSC will have the subspace detection property and a non-trivial output. Recall that we defined

$$r_\ell := \min_{i:x_i \in \mathcal{X}_\ell} r_{S_\ell}(\mathcal{Q}_{-i}^{(\ell)}).$$

$$\mu_\ell := \mu(\mathcal{X}_\ell).$$

We will also define $\mu := \max_\ell \mu_\ell$ and $r := \min_\ell r_\ell$.

Theorem 6.6 (Deterministic model criteria)**.** *Suppose that*

$$\delta \leq \frac{r - \mu}{5} \tag{6.4}$$

and λ lies in the non-empty interval

$$\frac{5}{2r + 3\mu} < \lambda < \frac{15}{2r + 8\mu}. \tag{6.5}$$

Then the subspace detection property with parameter λ will hold. Moreover, we are guaranteed that each c_i found in (5.3) *will be non-trivial.*

This theorem gives conditions that guarantee when c_i will not have any false positives. It then refines this to find λ for which the c_i will also be non-trivial. In fact, we will show more generally that it suffices to have

$$\delta \leq \frac{r_\ell - \mu_\ell}{5}$$

for all ℓ. When $\delta = 0$, this latter condition reduces to having $\mu_\ell < r_\ell$ for all ℓ, which is the same deterministic criteria as in [86].

[1]The results below all generalize to the case that Y is unnormalized. The results will depend on the gap between the largest and smallest norm of columns in Y.

We will refer to the condition in (6.4) as the **geometric separation condition**. We will show that under random model assumptions, the geometric separation condition will hold with high probability.

6.2.2 Random Model

In the random model, we fix dimensions d_ℓ of each subspace S_ℓ. We assume that S_ℓ is chosen uniformly at random among all subspaces of dimension d_ℓ and that the S_ℓ are chosen independently. Note that the collection of all d_ℓ-dimensional subspaces of $\mathbb{R}^n$ form a compact smooth manifold called the Grassmannian. The Grassmannian has an associated measure. Standard properties of the Grassmannian and its associated measure (referred to as the Haar measure) show that we can indeed sample uniformly from this space [65].

For each ℓ, we pick $N_\ell = \kappa_\ell d_\ell$ points y_i on the intersection of the unit sphere and S_ℓ. We then add a noise matrix Z. Our only assumption on Z is that for columns y_i and y_j drawn from distinct subspaces, z_i is independent to y_j. In other words, the noise added to the samples in one subspace is independent to the observations in other subspaces. As above, we define

$$\delta := \max_i \|z_i\|_2.$$

The theorems below give conditions on the noise under which the subspace detection property holds with high probability.

Theorem 6.7 (Random model criteria)**.** *There are absolute constants c_1, c_2 such that, if for all ℓ,*

$$d_\ell \leq \frac{c_1 \rho(\kappa_\ell)^2 \log(\kappa_\ell) n}{\log N} \tag{6.6}$$

and

$$\delta \leq c_2 \rho(\kappa_\ell) \sqrt{\frac{\log(\kappa_\ell)}{d_\ell}}. \tag{6.7}$$

then with probability at least

$$1 - \frac{2}{N} - \sum_{\ell=1}^{L} N_\ell e^{-\sqrt{\kappa_\ell} d_\ell}$$

the subspace detection property holds and the output of LS-SSC is non-trivial for all λ *satisfying*

$$\frac{10}{9}\sqrt{\frac{n}{24 \log N}} < \lambda < \frac{15}{6}\sqrt{\frac{n}{24 \log N}}.$$

Here, $\rho(\kappa)$ is a constant depending only on κ. It is the same constant $\rho(\kappa)$ as that in [86]. This work also shows that all κ sufficiently large we can take $\rho(\kappa)$ to be a constant. Moreover, in most reasonable scenarios, we can treat $\rho(\kappa_\ell)$ as a small constant.

It is also worth noting that the condition that $d_\ell \leq c_1 \rho(\kappa_\ell)^2 \log(\kappa_\ell) n / \log N$ is, up to constants, the same as the condition on d_ℓ required for subspace clustering to succeed without noise [86]. Therefore, for δ sufficiently small, we recover the sufficient condition in [86] for subspace detection under the same random model.

6.2.3 Missing Data Model

As in the random model, we assume that the L subspaces are chosen independently and uniformly at random, and that the points y_i are drawn randomly from the unit ball in S_ℓ. Our data matrix X satisfies $X = Y + Z$ where, $Z_{ij} = -Y_{ij}$ if we do not view Y_{ij} and zero otherwise. In other words, X is the entry-wise zero fill of Y in the missing entries.

We do not assume that we know the locations of the missing entries. A zero entry in X could be because we do not observe that entry, or because there is a zero in Y there. This allows LS-SSC to be used in more general settings such as with presence-only data.

For a vector v, let $\|v\|_0$ denote the number of non-zero entries in v. We assume that for all $y_i \in S_\ell$, $\|z_i\|_0 \leq m_\ell$. Our only assumptions on the missing data locations are that for any column y_i, the location of the missing entries in y_i are chosen independently from all columns of Y. In other words, the missing entry locations are chosen independently from Y. The following theorem gives conditions under which LS-SSC succeeds in this model.

Theorem 6.8 (Missing data criteria)**.** *There are absolute constants c_1, c_3 such that if for all ℓ*

$$d_\ell \leq \frac{c_1 \rho(\kappa_\ell)^2 \log(\kappa_\ell) n}{\log N}, \tag{6.8}$$

and the number of missing entries m_ℓ in any column drawn from S_ℓ satisfies

$$m_\ell \leq M_\ell \coloneqq c_3 \rho(\kappa_\ell)^2 \log(\kappa_\ell) \frac{n}{d_\ell},$$

then with probability at least

$$1 - \frac{2}{N} - \sum_{\ell=1}^{L} N_\ell e^{-\sqrt{\kappa_\ell} d_\ell} - 2 \sum_{\ell=1}^{L} N_\ell e^{-M_\ell/16},$$

the subspace detection property holds and the output of LS-SSC is non-trivial for all λ satisfying

$$\frac{10}{9} \sqrt{\frac{n}{24 \log N}} < \lambda < \frac{15}{6} \sqrt{\frac{n}{24 \log N}}.$$

Here, the constant c_1 is the same as in Theorem 6.7. This condition says that if the dimension d of our subspaces obey the same condition required for SSC without noise to succeed, then LS-SSC will succeed in the presence of $O(n/d)$ missing entries per column with high probability. If d is constant with respect to n, then this says that LS-SSC can tolerate a constant fraction of missing entries in each column.

6.3 Dual Certificates and the Deterministic Model

In this section we will analyze the primal and dual problems associated to LS-SSC. In particular, we will show that the existence of certain solutions to the associated dual problem will guarantee that the solutions to LS-SSC will satisfy the subspace detection property. This will allow us to prove Theorem 6.6 above.

6.3.1 Dual Certificates

Recall that the optimization problem in LS-SSC is equivalent to solving, for all i, $P(x_i, X_{-i}, \lambda)$. In order to guarantee that the solution (c, e) to these problems contains no false positives, we consider an idealized problem. That is, we analyze $P(x_i, X_{-i}^{(\ell)}, \lambda)$.

While we cannot solve this problem in practice, we show below that as long as the solution to this idealized problem and its dual satisfy certain conditions, the solution to $P(x_i, X_{-i}, \lambda)$ will satisfy the subspace detection property. This follows from similar techniques to those in [96]. The geometric separation condition guarantees the existence of these *dual certificates* to the subspace detection property.

We are interested in the *support* of the solution c^* of $P(x_i, X_{-i}, \lambda)$. We want c^* to have support only on the columns of X_{-i} coming from S_ℓ. In order to guarantee this, we prove the following lemma giving conditions that guarantee that c^* has the desired support.

Lemma 6.9. *Let $A \in \mathbb{R}^{n\times N}, x \in \mathbb{R}^n$ be such that there are vectors c, e, ν and sets $S \subseteq T \subseteq \{1, \ldots, N\}$ such that $e = x - Ac$, c has support S, and ν satisfies:*

1. *$A_S^T \nu = sgn(c_S)$*

2. $\nu = \lambda e$

3. $\|A^T_{T\cap S^c}\nu\|_\infty \leq 1$

4. $\|A^T_{T^c}\nu\|_\infty < 1$

Then any optimal solution (c^*, e^*) *to* $P(x, A, \lambda)$ *satisfies* $c^*_{T^c} = 0$.

Proof. Let (c^*, e^*) be a solution to $P_1(x, A, \lambda)$. Then we have:

$$\begin{aligned}
&\|c^*\|_1 + \frac{\lambda}{2}\|e^*\|_2^2 \\
&= \|c^*_S\|_1 + \|c^*_{T\cap S^c}\|_1 + \|c^*_{T^c}\|_1 + \frac{\lambda}{2}\|e^*\|_2^2 \\
&\geq \|c_S\|_1 + \langle \mathrm{sgn}(c_S), c^*_S - c_S\rangle + \|c^*_{T\cap S^c}\|_1 + \|c^*_{T^c}\|_1 + \frac{\lambda}{2}\|e^*\|_2^2. \qquad (6.9)
\end{aligned}$$

We now wish to find a lower bound for $\frac{\lambda}{2}\|e^*\|_2^2$ involving e. Define

$$f(e) = \lambda(-\frac{1}{2}e^Te + e^Te^*).$$

Then $f(e^*) = \frac{\lambda}{2}\|e^*\|_2^2$. Simple calculus shows that this is the maximum value of f and is only achieved by e^*. Therefore, for any other e, $f(e) \leq f(e^*)$. In particular this implies

$$\begin{aligned}
\frac{\lambda}{2}\|e^*\|_2^2 &= f(e^*) \\
&\geq f(e) \\
&= \lambda(-\frac{1}{2}e^Te + e^Te^*) \\
&= \frac{\lambda}{2}\|e\|_2^2 + \langle \lambda e, e^* - e\rangle.
\end{aligned}$$

Using this fact and Assumptions 1 and 2 on ν in (6.9), we have

$$\begin{aligned}
&\|c^*\|_1 + \frac{\lambda}{2}\|e^*\|_2^2 \\
&\geq \|c_S\|_1 + \langle \nu, A_S(c_S^* - c_S)\rangle + \|c^*_{T\cap S^c}\|_1 + \|c^*_{T^c}\|_1 + \frac{\lambda}{2}\|e\|_2^2 + \langle \nu, e^* - e\rangle \\
&\geq \|c_S\|_1 + \frac{\lambda}{2}\|e\|_2^2 + \|c^*_{T\cap S^c}\|_1 - \langle \nu, A_{T\cap S^c} c^*_{T\cap S^c}\rangle + \|c^*_{T^c}\|_1 - \langle \nu, A_{T^c} c^*_{T^c}\rangle \\
&\qquad + \langle \nu, A(c^* - c) + e^* - e\rangle. \qquad (6.10)
\end{aligned}$$

Since (c^*, e^*) and (c, e) are feasible, we have

$$\begin{aligned}
Ac^* + e^* = x = Ac + e \\
\implies A(c^* - c) + e^* - e = 0. \qquad (6.11)
\end{aligned}$$

Combining (6.10) and (6.11), and using the fact that c has support S so $c_S = c$, we have

$$\begin{aligned}
&\|c^*\|_1 + \frac{\lambda}{2}\|e^*\|_2^2 \\
&\geq \|c\|_1 + \frac{\lambda}{2}\|e\|_2^2 + \|c^*_{T\cap S^c}\|_1 - \langle \nu, A_{T\cap S^c} c^*_{T\cap S^c}\rangle + \|c^*_{T^c}\|_1 - \langle \nu, A_{T^c} c^*_{T^c}\rangle.. \qquad (6.12)
\end{aligned}$$

By Assumption 3 on ν, we have

$$\begin{aligned}
\langle \nu, A_{T\cap S^c} c^*_{T\cap S^c}\rangle &= \langle A^T_{T\cap S^c}\nu, c^*_{T\cap S^c}\rangle \\
&\leq \|A^T_{T\cap S^c}\nu\|_\infty \|c^*_{T\cap S^c}\|_1 \\
&\leq \|c^*_{T\cap S^c}\|_1. \qquad (6.13)
\end{aligned}$$

By simple norm properties, we have

$$\begin{aligned}
\|c^*_{T^c}\|_1 - \langle \nu, A_{T^c} c^*_{T^c}\rangle &= \|c^*_{T^c}\|_1 - \langle A^T_{T^c}\nu, c^*_{T^c}\rangle \\
&\geq \|c^*_{T^c}\|_1 - \|A^T_{T^c}\nu\|_\infty \|c^*_{T^c}\|_1 \\
&\geq (1 - \|A^T_{T^c}\nu\|_\infty)\|c^*_{T^c}\|_1 \qquad (6.14)
\end{aligned}$$

Combining (6.12), (6.13), and (6.14), we find

$$\|c^*\|_1 + \frac{\lambda}{2}\|e^*\|_1 \geq \|c\|_1 + \frac{\lambda}{2}\|e\|_1 + (1 - \|A_{T^c}^T \nu\|_\infty)\|c_{T^c}^*\|_1.$$

By Assumption 4 on ν, we know that $(1 - \|A_{T^c}^T \nu\|_\infty) > 0$. By optimality of c^*, e^* for $P(x, A, \lambda)$, this implies that $\|c_{T^c}^*\|_1 = 0$ and so $c_{T^c}^* = 0$. Therefore, c^* has support contained in T. □

Let $T \subseteq \{1, \ldots, N\}$ denote the set of columns that correspond to S_ℓ. We wish to guarantee that the solution (c^*, e^*) to $P(x_i, X_{-i}, \lambda)$ has c^* with support contained in T. By Lemma 6.9, it suffices to exhibit, for each i, vectors c_i, e_i, ν_i satisfying the conditions of this lemma when we take $A = X_{-i}$ and $x = x_i$.

Therefore, we wish to show that for all i, there are vectors c_i, e_i such that $X_{-i}c_i - x_i = e_i$ with c_i having support contained in T, and that satisfy conditions 1-4 in Lemma 6.9.

We will construct (c_i, e_i, ν_i) in the following way. Let (c, e) solve $P(x_i, X_{-i}^{(\ell)}, \lambda)$. To see that such a (c, e) exists, it suffices to show that the optimization problem is feasible. This follows from the fact that we can take any $\tilde{c}$ and let $\tilde{e} = x_i - X_{-i}^{(\ell)}\tilde{c}$. Note that since we have no inequality constraints, Slater's condition holds and so does strong duality. The dual is also strictly feasible since $\tilde{\nu} = 0$ will satisfy $\|(X_{-i}^{(\ell)})^T \nu\|_\infty < 1$.

We define c_i to be 0 in all columns outside of $X_{-i}^{(\ell)}$ and let it equal c in columns corresponding to $X_{-i}^{(\ell)}$. Let $e_i = e$. Note that $X_{-i}c_i - x_i = e_i$, so this is a feasible point of $P(x_i, X_{-i}, \lambda)$.

We let ν_i denote the dual vector to c guaranteed by strong duality. For ease of notation, we will now fix i and let ν equal ν_i. That is, ν is the solution to $D(x_i, X_{-i}^{(\ell)}, \lambda)$. In particular, ν satisfies

$$\|(X_{-i}^{(\ell)})^T \nu\|_\infty \leq 1.$$

Let S denote the support of c. This implies that for $A = X_{-i}$, we have

$$\|A_{T\cap S^c}^T \nu\|_\infty \leq 1.$$

Therefore, Assumption 3 of Lemma 6.9 holds. Complementary slackness implies the following conditions:

$$(X_{-i}^{(\ell)})_S^T \nu = \text{sgn}(c_S), \tag{6.15}$$

$$\nu = \lambda e. \tag{6.16}$$

Therefore, (c_i, e_i, ν) also satisfy Assumptions 1 and 2 of Lemma 6.9. The remaining assumption we must show is Assumption 4. Therefore, to prove the subspace detection property holds, it suffices to show that for each column x of X not in S_ℓ, the following condition satisfied:

$$|\langle x, \nu\rangle| < 1. \tag{6.17}$$

It therefore suffices to show that the conditions in Lemma 6.17 imply that $|\langle x, \nu\rangle| < 1$ for all $x \in \mathcal{X}\backslash\mathcal{X}^\ell$ and ν in the statement of the lemma.

Fix $x \in \mathcal{X}\backslash\mathcal{X}^\ell$, where $x = y + z$. Here, y is the true vector and z is the noise vector. We wish to give a condition under which $|\langle x, \nu\rangle| < 1$. Note that we have

$$|\langle x, \nu\rangle| \leq |\langle y, \nu\rangle| + |\langle z, \nu\rangle|. \tag{6.18}$$

We bound these two terms separately. For the first, note that

$$|\langle y, \nu\rangle| = \|\nu\|_2 \left|\langle y, \frac{\nu}{\|\nu\|_2}\rangle\right|. \tag{6.19}$$

By the definition of $\mu(\mathcal{X}^{(\ell)})$, we have

$$\left|\langle y, \frac{\nu}{\|\nu\|_2}\rangle\right| \leq \mu(\mathcal{X}^{(\ell)}). \tag{6.20}$$

By Cauchy-Schwarz, we have

$$|\langle z, \nu \rangle| \leq \delta \|\nu\|_2. \tag{6.21}$$

Combining this we have,

$$|\langle x, \nu \rangle| \leq (\mu(\mathcal{X}_\ell) + \delta)\|\nu\|_2. \tag{6.22}$$

Therefore, we have the following result.

Lemma 6.10. *Suppose that for all i and ℓ where (c_i, e_i) solve $P(x_i, X^{(\ell)}_{-i}, \lambda)$ and ν is the corresponding solution to $D(x_i, X^{(\ell)}_{-i}, \lambda)$, we have*

$$(\mu(\mathcal{X}_\ell) + \delta)\|\nu\|_2 < 1.$$

Then the subspace detection property with parameter λ holds.

In the next section we will bound $\|\nu\|_2$ in order to better utilize Lemma 6.10.

6.3.2 Bounding $\|\nu\|_2$

We now wish to bound $\|\nu\|_2$. We will do this by bounding the norm of its projection on to S_ℓ and the norm of its part that is orthogonal to S_ℓ.

Recall that the columns of $Y^{(\ell)}$ all lie in the subspace S_ℓ. Let Proj_{S_ℓ} denote the projection operator on to S_ℓ. We define

$$\nu_1 := \underset{S_\ell}{\text{Proj}}\, \nu.$$

$$\nu_2 := \nu - \nu_1.$$

We will bound $\|\nu_1\|_2, \|\nu_2\|_2$ separately and use the fact that by orthogonality, $\|\nu\|_2 = \|\nu_1\|_2 + \|\nu_2\|_2$. To bound $\|\nu_1\|_2$, we will require the following fact from convex geometry.

Lemma 6.11. *Let $\mathcal{Q}$ be the symmetrized convex hull of a set of points $\mathcal{Y} \subseteq \mathbb{R}^n$ spanning a subspace S of dimension d. For each $y \in \mathcal{Y}$, let $x = y + z$ for $z \in S$ with $\|z\|_2 \leq \delta$. Denote the collection of such x as $\mathcal{X}$ and let $\mathcal{T}$ denote its symmetrized convex hull. Suppose that $r_S(\mathcal{Q}) > \delta \geq 0$. Then*

$$r_S(\mathcal{T}) \geq r_S(\mathcal{Q}) - \delta.$$

Proof. Note that since each $y, z \in S$, each $x \in S$. Therefore, $\mathcal{Q}$ and $\mathcal{T}$ both lie in S. We can perform a change of coordinates that preserves all distances between points in S such that $\mathcal{Q}$ spans the first d dimensions of $\mathbb{R}^n$. Then note that the relative inradius $r_S(\mathcal{Q})$ equals the absolute inradius $r(\mathcal{Q})$ in $\mathbb{R}^d$. The same occurs for $\mathcal{T}$. It therefore suffices to show that for $\mathcal{Q}$ and $\mathcal{T} \subseteq \mathbb{R}^d$, $r(\mathcal{T}) \geq r(\mathcal{Q}) - \delta$.

Since $\mathcal{Q}$ and $\mathcal{T}$ are symmetric and convex, it suffices to show that the ball $\mathcal{B}$ of radius $r(\mathcal{Q}) - \delta$ centered at the origin is contained within $\mathcal{T}$. Consider a face F of $\mathcal{Q}$. There are vertices $y_1, \ldots, y_k$ such that

$$F = \left\{ \sum_{i=1}^{k} y_i w_i \middle| 0 \leq w_i \leq 1, \sum_{i=1}^{k} w_i = 1 \right\}.$$

Since F is on the boundary of $\mathcal{Q}$, every point $y \in F$ satisfies $\|y\|_2 \geq r(\mathcal{Q})$. Let $x_1, \ldots, x_k$ denote the points in $\mathcal{T}$ corresponding to the y_i (that is, $x_i = y_i + z_i$ for some z_i with $\|z_i\|_2 \leq \delta$). Define F' by

$$F' = \left\{ \sum_{i=1}^{k} x_i w_i \middle| 0 \leq w_i \leq 1, \sum_{i=1}^{k} w_i = 1 \right\}.$$

Now suppose $x \in F'$. Then we have

$$\begin{aligned}
\|x\|_2 &= \left\| \sum_{i=1}^{k} x_i w_i \right\|_2 \\
&= \left\| \sum_{i=1}^{k} y_i w_i + z_i w_i \right\|_2 \\
&\geq \left\| \sum_{i=1}^{k} y_i w_i \right\|_2 - \sum_{i=1}^{k} w_i \|z_i\|_2 \\
&\geq r(\mathcal{Q}) - \sum_{i=1}^{k} w_i \delta \\
&\geq r(\mathcal{Q}) - \delta.
\end{aligned}$$

Let S be the union of F' over all faces F of $\mathcal{Q}$. Then S is the boundary of some closed (not necessarily convex, possibly self-intersecting) polytope $\mathcal{P}$ containing the origin. Moreover, since every point in S has norm at least $r(\mathcal{Q}) - \delta$, we know that $\mathcal{P}$ contains $\mathcal{B}$. Since each vertex x_i of each F' is in $\mathcal{T}$ and $\mathcal{T}$ is convex, we know that $\mathcal{P} \subseteq \mathcal{T}$. Therefore,

$$r(\mathcal{T}) \geq r(\mathcal{P}) \geq r(\mathcal{Q}) - \delta.$$

□

We will also use the following lemma about the relation between the relative inradius and circumradius of a convex body. This lemma can be bound in [15].

Lemma 6.12 ([15])**.** *For any symmetric convex polytope $\mathcal{T}$,*

$$r(\mathcal{T}) R(\mathcal{T}^\circ) = 1.$$

We can then use this to derive the following lemma about the relative inradius of a symmetric convex polytope.

Lemma 6.13. *Suppose that $\mathcal{T}$ is a symmetric convex polytope that lies in a subspace S. Then*

$$r_S(\mathcal{T})R_S(\mathcal{T}^\circ \cap S) = 1.$$

Proof. Suppose $\mathcal{T} \subseteq \mathbb{R}^n$ and S has dimension d. After performing an appropriate distance-preserving change of coordinates, we can assume that S is the span of the elements $e_1, \ldots, e_d$ where e_i denotes the ith vector of the standard basis. Let $\pi : \mathbb{R}^n \to \mathbb{R}^d$ be the map that sends a vector $(x_1, \ldots, x_n)$ to $(x_1, \ldots, x_d)$. Let $\mathcal{T}' = \pi(\mathcal{T})$. Note that this is the same convex polytope as $\mathcal{T}$, but in $\mathbb{R}^d$ instead of $\mathbb{R}^n$. In particular, $r_S(\mathcal{T}) = r(\mathcal{T}')$.

Moreover, simple linear algebra shows that $\pi(\mathcal{T}^\circ) = \pi(\mathcal{T}^\circ \cap S) = (\mathcal{T}')^\circ$. In particular, $R_S(\mathcal{T}^\circ \cap S) = R((\mathcal{T}')^\circ)$. By Lemma 6.12, we then have

$$r_S(\mathcal{T})R_S(\mathcal{T}^\circ \cap S) = r(\mathcal{T}')R((\mathcal{T}')^\circ) = 1.$$

□

We can then show the following Lemma.

Lemma 6.14.

$$\|\nu_1\|_2 \leq \frac{1 + \delta\|\nu_2\|_2}{r_{S_\ell}(\mathcal{Q}_{-i}^{(\ell)}) - \delta}.$$

Proof. Since ν is a feasible point of $D(x_i, X_{-i}^{(\ell)}, \lambda)$, we know that

$$\|(X_{-i}^{(\ell)})^T \nu\|_\infty \leq 1.$$

In particular, for any column $x = y + z$ of $X_{-i}^{(\ell)}$ we have

$$\begin{aligned} |\langle x, \nu \rangle| &\leq |\langle y, \nu_1 \rangle + \langle y, \nu_2 \rangle + \langle z, \nu_1 \rangle + \langle z, \nu_2 \rangle| \\ &= |\langle y, \nu_1 \rangle + \langle \operatorname*{Proj}_{S_\ell} z, \nu_1 \rangle + \langle z, \nu_2 \rangle| \\ &\leq 1. \end{aligned}$$

Here we used the fact that $\nu_1 \in S_\ell$ while $\nu_2 \in S_\ell^\perp$. Since $y \in S_\ell$, we have $\operatorname{Proj}_{S_\ell} x = y + \operatorname{Proj}_{S_\ell} z$. Therefore,

$$|\langle \operatorname*{Proj}_{S_\ell} x, \nu_1 \rangle| \leq 1 + |\langle z, \nu_2 \rangle| \leq 1 + \delta \|\nu_2\|_2.$$

Therefore,

$$\left\| (Y_{-i}^{(\ell)} + \operatorname*{Proj}_{S_\ell} Z_{-i}^{(\ell)})^T \frac{\nu_1}{1 + \delta \|\nu_2\|_2} \right\| \leq 1. \tag{6.23}$$

Let $M = Y_{-i}^{(\ell)} + \operatorname{Proj}_{S_\ell} Z_{-i}^{(\ell)}$ and let $\mathcal{T} = \mathcal{SC}(M)$. Recall that this is the symmetrized convex hull of the columns of T. Then (6.24) implies that

$$\frac{\nu_1}{1 + \delta \|\nu_2\|_2} \in \mathcal{T}^\circ. \tag{6.24}$$

Since $\nu_1 \in S_\ell$, the circumradius of $\mathcal{T}^\circ \cap S_\ell$ relative to S_ℓ provides a bound on the norm of $\nu_1/(1 + \delta \|\nu_2\|_2)$. Therefore,

$$\|\nu_1\|_2 \leq R_{S_\ell}(\mathcal{T}^\circ \cap S_\ell)(1 + \delta \|\nu_2\|_2). \tag{6.25}$$

We now wish to bound $R_{S_\ell}(\mathcal{T}^\circ \cap S_\ell)$. By Lemma 6.13, we have

$$R_{S_\ell}(\mathcal{T}^\circ \cap S) = \frac{1}{r_{S_\ell}(\mathcal{T})}.$$

It now suffices to give a lower bound on $r_{S_\ell}(\mathcal{T})$. Recall that $\mathcal{T}$ is the symmetrized convex hull of the columns of $Y_{-i}^{(\ell)}$, perturbed by $\operatorname{Proj}_{S_\ell} Z_{-i}^{(\ell)}$. Since each column z has

norm at most δ, we can apply Lemma 6.11, which then implies

$$r_{S_\ell}(\mathcal{T}) \geq r_{S_\ell}(\mathcal{SC}(Y_{-i}^{(\ell)})) - \delta = r_{S_\ell}(\mathcal{Q}_{-i}^{(\ell)}) - \delta.$$

Therefore,

$$\|\nu_1\|_2 \leq \frac{1 + \delta\|\nu_2\|_2}{r(\mathcal{Q}_{-i}^{(\ell)}) - \delta}.$$

□

We bound $\|\nu_2\|_2$ in the following lemma.

Lemma 6.15.

$$\|\nu_2\|_2 \leq \lambda\delta\left(\frac{1}{r_{S_\ell}(\mathcal{Q}_{-i}^{(\ell)})} + 1\right).$$

Proof. Recall that ν is a solution of $D(x_i, X_{-i}^{(\ell)}, \lambda)$, and that by (6.16), we have $\nu = \lambda e_i$. Therefore, $\nu = \lambda(x_i - X_{-i}^{(\ell)}c_i)$. In the following, we will let c_{ij} denote the jth entry of c_i. Therefore,

$$\begin{aligned}
\|\nu_2\|_2 &= \|\operatorname*{Proj}_{S_\ell^\perp} \nu\|_2 \\
&= \lambda\|\operatorname*{Proj}_{S_\ell^\perp}(x_i - X_{-i}^{(\ell)}c_i)\|_2 \\
&= \lambda\|\operatorname*{Proj}_{S_\ell^\perp} z_i + \operatorname*{Proj}_{S_\ell^\perp} Z_{-i}^{(\ell)}c_i\|_2 \\
&\leq \lambda\|\operatorname*{Proj}_{S_\ell^\perp} z_i\|_2 + \lambda\|\operatorname*{Proj}_{S_\ell^\perp} Z_{-i}^{(\ell)}c_i\|_2 \\
&\leq \lambda\delta + \lambda\left(\sum_j |c_{ij}|\|\operatorname*{Proj}_{S_\ell^\perp} z_j\|_2\right) \\
&\leq \lambda\delta + \lambda\sum_j |c_{ij}|\delta \\
&\leq \lambda\delta(1 + \|c_i\|_1).
\end{aligned}$$

We wish to bound $\|c_i\|_1$. Note that for any other feasible point $(\hat{c}_i, \hat{e}_i)$ of $P(x_i, X_{-i}, \lambda)$, by optimality we must have

$$\|c_i\|_1 + \frac{\lambda}{2}\|e_i\|_2^2 \leq \|\hat{c}_i\|_1 + \frac{\lambda}{2}\|\hat{e}_i\|_2^2.$$

By (6.16), we have $\lambda e_i = \nu$. Therefore,

$$\frac{\lambda}{2}\|\hat{e}_i\|_2^2 = \frac{1}{2\lambda}\|\nu\|_2^2 \geq \frac{1}{2\lambda}\|\nu_2\|_2^2.$$

Therefore,

$$\begin{aligned} \|\nu_2\|_2 &\leq \lambda\delta(1 + \|c_i\|_1) \\ &\leq \lambda\delta + \lambda\delta\left(\|\hat{c}_i\|_1 + \frac{\lambda}{2}\|\hat{e}_i\|_2^2 - \frac{1}{2\lambda}\|\nu_2\|_2^2\right) \\ \implies \|\nu_2\|_2 + \frac{\delta}{2}\|\nu_2\|_2^2 &\leq \lambda\delta + \lambda\delta\left(\|\hat{c}_i\|_1 + \frac{\lambda}{2}\|\hat{e}_i\|_2^2\right). \end{aligned} \tag{6.26}$$

We now wish to construct $(\hat{c}_i, \hat{e}_i)$ such that we can bound $\|\hat{c}_i\|_1 + \frac{\lambda}{2}\|\hat{e}_i\|_2^2$. We will do this by letting $\hat{c}_i$ be the solution to

$$\min_c \|c\|_1 \text{ s.t. } y_i = Y_{-i}^{(\ell)} c. \tag{6.27}$$

To make this point feasible, we define

$$\hat{e}_i = z_i - Z_{-i}^{(\ell)}\hat{c}_i.$$

Since we assume that Y is self-expressive, we know that the optimization problem used to define $\hat{c}_i$ has a non-empty feasible set. Therefore, it satisfies strong duality. Let $\hat{\nu}$ be the corresponding solution to the dual problem to (6.27) of smallest norm, that is $\hat{\nu}$ solves

$$\max_\nu \langle y_i, \nu\rangle \text{ s.t. } \|(Y_{-i}^{(\ell)})^T\nu\|_\infty \leq 1. \tag{6.28}$$

Note that if ν is optimal in (6.28) then so is $\mathrm{Proj}_{S_\ell}\nu$ as this does not alter the objective nor does it violate any constraint, as y_i and the columns of $Y_{-i}^{(\ell)}$ all lie in S_ℓ. Therefore, if we take $\hat{\nu}$ to be the solution of the smallest norm, we must have $\hat{\nu} \in S_\ell$. We also know by (6.28) that $\hat{\nu} \in (\mathcal{Q}_{-i}^{(\ell)})^\circ$. Therefore,

$$\hat{\nu} \in (\mathcal{Q}_{-i}^{(\ell)})^\circ \cap S_\ell.$$

By Lemma 6.13,

$$\begin{aligned}\|\hat{\nu}\|_2 &\leq R_{S_\ell}((\mathcal{Q}_{-i}^{(\ell)})^\circ \cap S_\ell) \\ &= \frac{1}{r_{S_\ell}(\mathcal{Q}_{-i}^{(\ell)})}.\end{aligned}$$

By strong duality, we have

$$\|\hat{c}_i\|_1 = \langle y_i, \hat{\nu}\rangle \leq \|\hat{\nu}\|_2\|y_i\|_2 \leq \frac{1}{r_{S_\ell}(\mathcal{Q}_{-i}^{(\ell)})}. \tag{6.29}$$

We can bound $\|\hat{e}_i\|_2^2$ by using the fact that

$$\|\hat{e}_i\|_2 \leq \|z_i\|_2 + \sum_j |\hat{c}_{ij}|\|z_j\| \leq \delta(1 + \|\hat{c}_1\|_1). \tag{6.30}$$

Plugging (6.30) into (6.26) we get

$$\|\nu_2\|_2 + \frac{\delta}{2}\|\nu_2\|_2^2 \leq \lambda\delta(1 + \|\hat{c}_i\|_1) + \frac{\delta}{2}\left(\lambda\delta(1 + \|\hat{c}_i\|_1)\right)^2. \tag{6.31}$$

Note that the function $f(x) = x + \frac{\delta}{2}x^2$ monotonically increases for $x \geq 0$. Note that (6.31) is equivalent to

$$f(\|\nu_2\|_2) \leq f\big(\lambda\delta(1 + \|\hat{c}_i\|_1)\big).$$

By (6.29) we have

$$\|\nu_2\|_2 \leq \lambda\delta(1 + \|\hat{c}_i\|_1) \leq \lambda\delta\left(\frac{1}{r_{S_\ell}(\mathcal{Q}_{-i}^{(\ell)})} + 1\right).$$

□

Combining the bounds on $\|\nu_1\|_2, \|\nu_2\|_2$ we get the following lemma.

Lemma 6.16.

$$\|\nu\|_2 \leq \frac{1+\lambda\delta^2\left(\frac{1}{r_{S_\ell}(\mathcal{Q}_{-i}^{(\ell)})}+1\right)}{r_{S_\ell}(\mathcal{Q}_{-i}^{(\ell)})-\delta}+\lambda\delta\left(\frac{1}{r_{S_\ell}(\mathcal{Q}_{-i}^{(\ell)})}+1\right).$$

6.3.3 Towards a Deterministic Criteria

We can now make the first step towards a deterministic criteria for the success of LS-SSC. We will show the following lemma concerning when the subspace detection property holds.

Lemma 6.17. *Suppose that for all ℓ we have*

$$2\lambda\delta < \frac{r_\ell - \mu_\ell - 2\delta}{\mu_\ell + \delta}. \tag{6.32}$$

Then the subspace detection property with parameter λ holds.

Proof. By Lemma 6.10, it suffices to show that for all i and ℓ such that $y_i \in S_\ell$, we have

$$(\mu(\mathcal{X}_\ell)+\delta)\|\nu\|_2 < 1. \tag{6.33}$$

By Lemma 6.16, we have

$$(\mu(\mathcal{X}_\ell)+\delta)\|\nu\|_2 \leq (\mu(\mathcal{X}_\ell)+\delta)\left(\frac{1+\lambda\delta^2\left(\frac{1}{r_{S_\ell}(\mathcal{Q}_{-i}^{(\ell)})}+1\right)}{r_{S_\ell}(\mathcal{Q}_{-i}^{(\ell)})-\delta}+\lambda\delta\left(\frac{1}{r_{S_\ell}(\mathcal{Q}_{-i}^{(\ell)})}+1\right)\right).$$

Rearranging, (6.33) holds if

$$\lambda\delta^2\left(\frac{1}{r_{S_\ell}(\mathcal{Q}_{-i}^{(\ell)})}+1\right)+\lambda\delta\left(\frac{1}{r_{S_\ell}(\mathcal{Q}_{-i}^{(\ell)})}+1\right)(r_{S_\ell}(\mathcal{Q}_{-i}^{(\ell)})-\delta) < \frac{r_{S_\ell}(\mathcal{Q}_{-i}^{(\ell)})-\delta}{\mu(\mathcal{X}_\ell)+\delta}-1.$$

Manipulating further, this is equivalent to

$$\lambda\delta\left(\frac{1}{r_{S_\ell}(\mathcal{Q}_{-i}^{(\ell)})}+1\right)(\delta+r_{S_\ell}(\mathcal{Q}_{-i}^{(\ell)})-\delta) < \frac{r_{S_\ell}(\mathcal{Q}_{-i}^{(\ell)})-\mu(\mathcal{X}_\ell)-2\delta}{\mu(\mathcal{X}_\ell)+\delta}$$

$$\iff \quad \lambda\delta(1+r_{S_\ell}(\mathcal{Q}_{-i}^{(\ell)})) < \frac{r_{S_\ell}(\mathcal{Q}_{-i}^{(\ell)})-\mu(\mathcal{X}_\ell)-2\delta}{\mu(\mathcal{X}_\ell)+\delta}.$$

Note that since $r(\mathcal{Q}_{-i}^{(\ell)}) \leq 1$, this is satisfied if

$$2\lambda\delta < \frac{r_{S_\ell}(\mathcal{Q}_{-i}^{(\ell)})-\mu(\mathcal{X}_\ell)-2\delta}{\mu(\mathcal{X}_\ell)+\delta}.$$

Since the right-hand side decreases as $r_{S_\ell}(\mathcal{Q}_{-i}^{(\ell)})$ decreases, it suffices to satisfy this condition for i minimizing $r_{S_\ell}(\mathcal{Q}_{-i}^{(\ell)})$, which is how we defined r_ℓ. Also using the notation $\mu_\ell = \mu(\mathcal{X}_\ell)$, the subspace detection property holds if for all ℓ,

$$2\lambda\delta < \frac{r_\ell-\mu_\ell-2\delta}{\mu_\ell+\delta}.$$

□

Note that while this guarantees the subspace detection property, it does not yet guarantee that the output c will be non-trivial. We will show how to set λ appropriately to avoid this scenario in the following.

6.3.4 Finding Admissible λ

In this section, we prove a lemma that gives a bound on λ for which the output vector c of $P(x_i, X_{-i}, \lambda)$ is non-trivial. We say that such λ are *admissible*. This lemma was first stated in [96], though their proof leaves out a few minor details. For completeness, we restate the lemma and provide a detailed proof.

Lemma 6.18. *If for all ℓ,*

$$\lambda > \frac{1}{r_\ell - 2\delta - \delta^2},$$

then the solution (c, e) to $P(x_i, X_{-i}, \lambda)$ satisfies $c \neq 0$.

Proof. Suppose that the solution (c, e) to $P(x_i, X_{-i}, \lambda)$ satisfies $c = 0$. Then $e = x_i - X_i c = x_i$. Strong duality guarantees a dual vector ν that is feasible for $D(x_i, X_{-i}, \lambda)$. We will show that for λ large enough, the vector $(0, x_i)$ has dual ν that does not satisfy the condition $\|X_{-i}^T \nu\|_\infty \leq 1$.

Assume otherwise. By complementary slackness, $\nu = \lambda e = \lambda x_i$. Therefore,

$$\|X_{-i}^T \nu\|_\infty = \lambda \max_{j \neq i} |\langle x_j, x_i \rangle|.$$

Therefore, for any $j \neq i$ we then have

$$\begin{aligned}
\|X_{-i}^T \nu\|_\infty &\geq \lambda |\langle x_j, x_i \rangle| \\
&= \lambda \Big| \langle y_j, y_i \rangle + \langle y_j, z_i \rangle + \langle z_j, y_i \rangle + \langle z_j, z_i \rangle \Big| \\
&\geq \lambda \Big(|\langle y_j, y_i \rangle| - |\langle y_j, z_i \rangle| - |\langle z_j, y_i \rangle| - |\langle z_j, z_i \rangle| \Big) \\
&\geq \lambda \Big(|\langle y_j, y_i \rangle| - 2\delta - \delta^2 \Big).
\end{aligned}$$

Therefore,

$$\|X_{-i}^T \nu\|_\infty \geq \lambda \Big(\|Y_{-i}^T y_i\|_\infty - 2\delta - \delta^2 \Big).$$

Restricting to points lying in S_ℓ, this implies

$$\|X_{-i}^T \nu\|_\infty \geq \lambda \Big(\|(Y_{-i}^{(\ell)})^T y_i\|_\infty - 2\delta - \delta^2 \Big).$$

To bound this below, we require the following fact from convex analysis.

Lemma 6.19. *Suppose that $\mathcal{Q}$ is the symmetrized convex hull of a set of points $\{y_1, \ldots, y_M\}$ lying in a subspace S. Then for any unit vector $u \in S$ and any y_i,*

$$\max_i |\langle y_i, u\rangle| \geq r_S(\mathcal{Q}).$$

Proof. Using the same trick as in the beginning of the proof of Lemma 6.11, it suffices to show that for $\mathcal{Q} = \mathcal{SC}(\{y_1, \ldots, y_M\}) \subseteq \mathbb{R}^n$ and any unit vector $u \in \mathbb{R}^n$,

$$\max_i |\langle y_i, u\rangle| \geq r(\mathcal{Q}).$$

Fix some unit vector u. Let $\partial\mathcal{Q}$ denote the boundary of $\mathcal{Q}$ Let v be the point on $\partial\mathcal{Q}$ in the direction of u. Note that if no such vector exists, then $r(\mathcal{Q}) = 0$ necessarily, in which case the result trivially holds. Since $\mathcal{Q}$ is a symmetric convex polytope, v lies on some face F of $\mathcal{Q}$. Without loss of generality, we can assume that for some $m \leq M$,

$$F = \left\{ \sum_{i=1}^{m} y_i w_i | 0 \leq w_i \leq 1, \sum_{i=1}^{m} w_i = 1 \right\}.$$

Define

$$k = \operatorname*{argmax}_{1 \leq j \leq m} |\langle y_j, v\rangle|.$$

Therefore,

$$|\langle y_k, u\rangle| = \frac{|\langle y_k, v\rangle|}{\|v\|_2}.$$

We wish to show that $|\langle y_k, u\rangle| \geq \|v\|_2$. This is equivalent to showing

$$|\langle y_k, v\rangle| \geq \|v\|_2^2.$$

Note that we have

$$\begin{aligned}\|v\|_2^2 &= v^T v \\ &= \sum_{i=1}^{m} w_i \langle y_i, v\rangle \\ &\leq \sum_{i=1}^{m} w_i \max_{1\leq j\leq m} |\langle y_j, v\rangle| \\ &= \sum_{i=1}^{m} w_i |\langle y_k, v\rangle| \\ &= |\langle y_k, v\rangle|.\end{aligned}$$

Therefore, for any unit vector u pointing in the direction of a face F, there is some vertex y_k of F and some point $v \in F$ such that

$$|\langle y_k, u\rangle| \geq \|v\|_2.$$

Taking a supremum of the left-hand side over all faces F and all vertices of said faces, and taking an infimum right-hand side over all points on the boundary of $\mathcal{Q}$, we have

$$\max_{1\leq i\leq M} |\langle y_i, u\rangle| \geq \min_{v\in\partial\mathcal{Q}} \|v\|_2 = r(\mathcal{Q}).$$

□

Therefore, $\|(Y_{-i}^{(\ell)})^T y_i\|_\infty \geq r_{S_\ell}(\mathcal{Q}_{-i}^{(\ell)})$, so

$$\|X_{-i}^T \nu\|_\infty \geq \lambda(r_{S_\ell}(\mathcal{Q}_{-i}^{(\ell)}) - 2\delta - \delta^2).$$

Taking the minimum of $r_{S_\ell}(\mathcal{Q}_{-i}^{(\ell)})$ over all y_i lying in S_ℓ, we have

$$\|X_{-i}^T \nu\|_\infty \geq \lambda(r_\ell - 2\delta - \delta^2).$$

Since $\|X_{-i}^T \nu\|_\infty \leq 1$ for feasible ν, if

$$\frac{1}{r_\ell - 2\delta - \delta^2} < \lambda \tag{6.34}$$

then $(0, x_i)$ cannot be optimal $P(x_i, X_{-i}, \lambda)$, as the corresponding $\nu = \lambda x_i$ satisfies $\|X_{-i}^T \nu\|_\infty > 1$. □

6.3.5 Proof of Theorem 6.6

For posterity, we write the statement of the theorem here again. Recall that we have

$$r := \min_\ell r_\ell.$$

$$\mu := \max_\ell \mu_\ell.$$

Theorem 6.20 (Deterministic model criteria)**.** *Suppose that*

$$\delta \leq \frac{r - \mu}{5} \tag{6.35}$$

and λ lies in the non-empty interval

$$\frac{5}{2r + 3\mu} < \lambda < \frac{15}{2r + 8\mu}. \tag{6.36}$$

Then the subspace detection property with parameter λ will hold. Moreover, we are guaranteed that each c_i found in (5.3) *will be non-trivial.*

Proof. By Lemma 6.17 and Lemma 6.18, the desired condition will hold if for all ℓ,

$$2\lambda\delta < \frac{r_\ell - \mu_\ell - 2\delta}{\mu_\ell + \delta} \tag{6.37}$$

and

$$\frac{1}{r_\ell - 2\delta - \delta^2} < \lambda. \tag{6.38}$$

Note that the right-hand side of (6.37) decreases as r_ℓ decreases and as μ_ℓ increases. Therefore, we get a sufficient condition by replacing r_ℓ by r and μ_ℓ by μ. Further rearranging, (6.37) becomes

$$\lambda < \frac{\frac{r-\mu}{\delta} - 2}{2(\mu + \delta)}. \tag{6.39}$$

Assuming that $\delta \leq \frac{r-\mu}{5}$, (6.39) holds if

$$\lambda < \frac{3}{2}\frac{1}{\mu + \frac{r-\mu}{5}} = \frac{15}{2r + 8\mu}. \tag{6.40}$$

For the lower bound on λ, since (6.35) implies $\delta \leq r_\ell \leq 1$, a sufficient condition for (6.38) to hold is that for all ℓ

$$\frac{1}{r_\ell - 3\delta} < \lambda. \tag{6.41}$$

It suffices to show that this holds for $r = \min_\ell r_\ell$. Again using the fact that $\delta \leq \frac{r-\mu}{5}$, this holds if

$$\lambda > \frac{1}{r - \frac{3}{5}(r-\mu)} = \frac{5}{2r + 3\mu}. \tag{6.42}$$

Note that for (6.40) and (6.42) can hold simultaneously if

$$\frac{5}{2r + 3\mu} < \frac{15}{2r + 8\mu}$$

$$\iff 10r + 40\mu < 30r + 45\mu.$$

This holds for $0 \leq \mu < r \leq 1$, completing the proof. □

6.4 High-Dimensional Probability and the Random Model

We now assume the conditions of the random model. We wish to show that under some condition on the noise level δ, the subspace detection property holds with high

probability. Recall that we wish to show the following theorem.

Theorem 6.21 (Random model criteria)**.** *There are absolute constants c_1, c_2 such that, if for all ℓ,*

$$d_\ell \leq \frac{c_1 \rho(\kappa_\ell)^2 \log(\kappa_\ell) n}{\log N} \tag{6.43}$$

and

$$\delta \leq c_2 \rho(\kappa_\ell) \sqrt{\frac{\log(\kappa_\ell)}{d_\ell}}. \tag{6.44}$$

then with probability at least

$$1 - \frac{2}{N} - \sum_{\ell=1}^{L} N_\ell e^{-\sqrt{\kappa_\ell} d_\ell}$$

the subspace detection property holds and the output of LS-SSC is non-trivial for all λ satisfying

$$\frac{10}{9}\sqrt{\frac{n}{24 \log N}} < \lambda < \frac{15}{6}\sqrt{\frac{n}{24 \log N}}.$$

Proof. We want to show that with high probability, the condition of Theorem 6.6 holds. In particular, we want to show that with high probability, we have

$$\delta \leq \frac{r - \mu}{5}.$$

To bound the right-hand side from below, we need a lower bound on r and an upper bound on μ. As previously discussed, the following lemma was proved in [86].

Lemma 6.22 ([86])**.** *For all $\beta \in [0, 1]$,*

$$\mathbb{P}\left(r_\ell \geq \rho(\kappa_\ell) \sqrt{\frac{\beta \log(\kappa_\ell)}{d_\ell}} \right) \leq \exp(-d_\ell^\beta N_\ell^{1-\beta}).$$

Recall that $\frac{N_\ell}{d_\ell} = \kappa_\ell$. Therefore, $d_\ell N_\ell = \kappa_\ell d_\ell^2$. Selecting $\beta = \frac{1}{2}$ and taking a union bound over all (i, ℓ) such that $y_i \in S_\ell$, we get the following lemma:

Lemma 6.23.

$$\mathbb{P}\left(\forall(i,\ell)\ s.t.\ y_i \in S_\ell :\ r(\mathcal{Q}_{-i}^{(\ell)}) \geq \frac{\rho(\kappa_\ell)\sqrt{\log(\kappa_\ell)}}{\sqrt{2d_\ell}}\right) \geq 1 - \sum_{\ell=1}^{L} N_\ell e^{-\sqrt{\kappa_\ell} d_\ell}.$$

Note that this lemma is equivalent to saying that

$$\mathbb{P}\left(r \geq \min_\ell d\frac{\rho(\kappa_\ell)\sqrt{\log(\kappa_\ell)}}{\sqrt{2d_\ell}}\right) \geq 1 - \sum_{\ell=1}^{L} N_\ell e^{-\sqrt{\kappa_\ell} d_\ell}.$$

Next, recall that we define μ_ℓ by

$$\mu_\ell = \max_{\substack{y \in \mathcal{Y}\backslash\mathcal{Y}^{(\ell)} \\ 1 \leq i \leq N_\ell}} |\langle v_i^{(\ell)}, y\rangle|. \tag{6.45}$$

Here, $v_i^{(\ell)}$ is a unit vector by definition. Fix $y \in \mathcal{Y}\backslash\mathcal{Y}^{(\ell)}$ and some vector y_i drawn from S_ℓ. Then the dual direction v_i is a unit vector depending only on the samples drawn from S_ℓ. In particular, y is drawn independently from these samples therefore y and v_i are independent. We also know that y has marginal distribution that is uniform on the unit sphere. This follows from the fact that the subspace S_j from which y is drawn is selected uniformly among all d_j-dimensional subspaces, and y is selected uniformly at random from the unit ball in S_j. We can therefore use the following consequence of well-known results concerning spherical cap densities.

Lemma 6.24 ([6])**.** *Let y be a vector uniformly distributed on the unit sphere S^{n-1} and let a be a fixed unit vector on S^{n-1}. Then for any $\epsilon > 0$,*

$$\mathbb{P}\left(|\langle a, y\rangle| > \epsilon\right) \leq 2\exp\left(-\frac{n\epsilon^2}{2}\right).$$

Applying Lemma 6.24 with $\epsilon = \sqrt{6\log N/n}$, $a = v_i^{(\ell)}$ and the y above, we get:

$$\mathbb{P}\left(|\langle v_i^{(\ell)}, y\rangle| > \sqrt{\frac{6\log N}{n}}\right) \leq \frac{2}{N^3}. \tag{6.46}$$

Taking a union bound of (6.46) over all such y and pairs (i,ℓ) with $y_i \in S_\ell$, we get derive the following Lemma.

Lemma 6.25.

$$\mathbb{P}\left(\mu(\mathcal{X}_\ell) \leq \sqrt{\frac{6\log N}{n}}\ ,\ \forall \ell\right) \geq 1 - \frac{2}{N}.$$

In particular, this implies

$$\mathbb{P}\left(\mu \leq \sqrt{\frac{6\log N}{n}}\right) \geq 1 - \frac{2}{N}.$$

By the union bound, Lemmas 6.23 and 6.25 show that with probability at least $1 - \frac{2}{N} - \sum_{\ell=1}^{L} N_\ell e^{-\sqrt{\kappa_\ell} d_\ell}$, we have that for all ℓ,

$$r_\ell \geq \frac{\rho(\kappa_\ell)\sqrt{\log(\kappa_\ell)}}{\sqrt{2d_\ell}} \quad , \quad \mu_\ell \leq \sqrt{\frac{6\log N}{n}}.$$

Assume that for all ℓ,

$$d_\ell \leq \frac{\rho(\kappa_\ell)^2 \log(\kappa_\ell)}{48\log N} n. \tag{6.47}$$

So, with probability at least $1 - \frac{2}{N} - \sum_{\ell=1}^{L} N_\ell e^{-\sqrt{\kappa_\ell} d_\ell}$,

$$r_\ell \geq \frac{\rho(\kappa_\ell)\sqrt{\log(\kappa_\ell)}}{\sqrt{2d_\ell}} \geq \sqrt{\frac{24\log N}{n}} \geq 2\mu_\ell.$$

In particular, this implies that with the same probability

$$r \geq \sqrt{\frac{24\log N}{n}} \geq 2\mu.$$

Therefore, with this same probability

$$r - \mu \geq \frac{r}{2}$$

and so

$$\frac{r-\mu}{5} \geq \frac{\rho(\kappa_\ell)}{10\sqrt{2}} \frac{\sqrt{\log(\kappa_\ell)}}{\sqrt{d_\ell}}. \tag{6.48}$$

If we require

$$\delta \leq \frac{\rho(\kappa_\ell)}{10\sqrt{2}} \frac{\sqrt{\log(\kappa_\ell)}}{\sqrt{d_\ell}},$$

then with probability at least $1-\frac{2}{N}-\sum_{\ell=1}^{L} N_\ell e^{-\sqrt{\kappa_\ell} d_\ell}$, the geometric separation condition and therefore the subspace detection property will hold. Taking $c_1 = \frac{1}{48}, c_2 = \frac{1}{10\sqrt{2}}$, we get conditions (6.43) and (6.44).

By the same reasoning as in the proof of 6.6, we can derive the same interval of λ for which the subspace detection property will hold and the output of LS-SSC will be non-trivial. We showed in the proof of Theorem 6.6 that this will occur as long as

$$\frac{1}{r_\ell - 3\delta} < \lambda < \frac{\frac{r_\ell - \mu_\ell}{\delta} - 2}{2(\mu_\ell + \delta)} \tag{6.49}$$

holds for all ℓ. Above, we showed that with probability at least $1-\frac{2}{N}-\sum_{\ell=1}^{L} N_\ell e^{-\sqrt{\kappa_\ell} d_\ell}$, under the assumption on d_ℓ, we have that for all ℓ,

$$\begin{aligned} r_\ell &\geq \sqrt{\frac{24 \log N}{n}} \\ \mu_\ell &\leq \sqrt{\frac{6 \log N}{n}} \\ \delta &\leq \frac{1}{5}\sqrt{\frac{6 \log N}{n}} \end{aligned}$$

Note that if we decrease r_ℓ, and increase μ_ℓ, δ, the interval in (6.49) gets smaller (ie. it is contained by the previous interval). Using our bound on r_ℓ, μ_ℓ, δ above and plugging into 6.49, we find that with the same probability above, the subspace detection property with parameter λ will hold and the output of LS-SSC will be non-trivial as long as

$$\frac{10}{9}\sqrt{\frac{n}{24 \log N}} < \lambda < \frac{15}{6}\sqrt{\frac{n}{24 \log N}}. \tag{6.50}$$

This is a non-empty interval of λ for which LS-SSC has the subspace detection property and has non-trivial output, with the given probability above. $\square$

6.5 Random Projections and Missing Data

As noted above, clustering with missing data is a special case of subspace clustering with additive noise. Let $X = Y + Z$ where each entry Z_{ij} either equals $-Y_{ij}$ or 0. If the number of missing entries is not too large, then the corruption matrix Z is relatively sparse.

Recall that we assume that in each column coming from S_ℓ, we have at most m_ℓ missing entries. We make no assumptions on how these missing entries are selected except that the missing locations are chosen independently from the observations. We wish to show the following theorem.

Theorem 6.26 (Missing data criteria)**.** *There are absolute constants c_1, c_3 such that if for all ℓ*

$$d_\ell \leq \frac{c_1 \rho(\kappa_\ell)^2 \log(\kappa_\ell) n}{\log N}, \tag{6.51}$$

and the number of missing entries m_ℓ in any column drawn from S_ℓ satisfies

$$m_\ell \leq M_\ell \coloneqq c_3 \rho(\kappa_\ell)^2 \log(\kappa_\ell) \frac{n}{d_\ell},$$

then with probability at least

$$1 - \frac{2}{N} - \sum_{\ell=1}^{L} N_\ell e^{-\sqrt{\kappa_\ell} d_\ell} - 2 \sum_{\ell=1}^{L} N_\ell e^{-M_\ell/16},$$

the subspace detection property holds and the output of LS-SSC is non-trivial for all λ satisfying

$$\frac{10}{9} \sqrt{\frac{n}{24 \log N}} < \lambda < \frac{15}{6} \sqrt{\frac{n}{24 \log N}}.$$

Note that c_1 is the same constant as in Theorem 6.7.

It suffices to find a condition on m_ℓ such that the assumptions of Theorem 6.7 hold. To do this, we have to control $\|z\|_2$ for each column z of Z. In the missing data model, z is the negative of the projection of a column y of Y on to m coordinates. We can then use the following standard result about the effect of projections on ℓ_2 norms. For one reference (among many), see [88].

Lemma 6.27. *Let q be a rotation-invariant probability measure on the unit sphere S^{n-1} and let $U \subseteq \mathbb{R}^n$ be any m-dimensional subspace. For $x \in S^{n-1}$, let $P_U(x)$ denote the orthogonal projection of x on to U. Then for any $0 < \epsilon < 1$,*

$$q\left(\left\{x \in S^{n-1} : \|P_U(x)\|_2 \geq \frac{1}{1-\epsilon}\sqrt{\frac{m}{n}}\right\}\right) \leq 2e^{-\epsilon^2 m/2}.$$

This allows us to bound $\|z\|_2$ in terms of the number of missing entries m and the ambient dimension n of the data. If we select $m = O(n/d)$, where d is the dimension of the subspace containing y, then with high probability the conditions of Theorem 6.7 will hold for LS-SSC. The details are in the following proof.

Proof of Theorem 6.26. Note that the required condition on d_ℓ in (6.51) is the same as in Theorem 6.7. Therefore, to prove Theorem 6.26, it suffices to find conditions on the number of missing entries such that the required bound on δ in Theorem 6.7 holds. Let c_2 be the absolute constant from Theorem 6.7. It suffices to show that for each column z of Z, we have

$$\|z\|_2 \leq c_2 \rho(\kappa_\ell)\sqrt{\frac{\log(\kappa_\ell)}{d_\ell}}.$$

For each ℓ, define

$$M_\ell := \frac{c_2^2(1-\epsilon)^2\rho(\kappa_\ell)^2\log(\kappa_\ell)n}{d_\ell}.$$

Suppose that for all ℓ,

$$d_\ell \leq \frac{c_1 \rho(\kappa_\ell)^2 \log(\kappa_\ell) n}{\log N}$$

and

$$m_\ell \leq M_\ell.$$

Fix some column y in Y coming from S_ℓ. Since S_ℓ is chosen uniformly at random from all d_ℓ-dimensional subspaces and y is chosen uniformly on $S^{n-1} \cap S_\ell$, y has a marginal distribution that is rotational-invariant on the unit sphere. Let $\{i_1, \ldots, i_{m_\ell}\}$ denote the locations of the missing entries of y. Without loss of generality, we can assume that $m_\ell = M_\ell$, as decreasing the number of missing entries only decreases $\|z\|_2$, which we wish to bound from above.

Let U be the span of $\{e_{i_1}, \ldots, e_{i_{m_\ell}}\}$. Since U is chosen independently to Y, we can consider U as fixed with respect to y. Then $z = -P_U(y)$. By Lemma 6.27, for any $\epsilon > 0$, with probability at least $1 - 2e^{-\epsilon^2(1-\epsilon)M_\ell/4}$ we have

$$\begin{aligned}
\|z\|_2 &\leq (1-\epsilon)^{-1}\sqrt{\frac{m_\ell}{n}} \\
&\leq (1-\epsilon)^{-1}\sqrt{\frac{M_\ell}{n}} \\
&= c_2 \rho(\kappa_\ell)\sqrt{\frac{\log(\kappa_\ell)}{d_\ell}}.
\end{aligned}$$

Taking a union bound, this holds for all columns z of Z with probability at least

$$1 - 2\sum_{\ell=1}^{L} N_\ell e^{-M_\ell/16}.$$

Therefore, the conditions of Theorem 6.7 hold with at least this probability. Taking $\epsilon = \frac{1}{2}$, letting $c_3 = c_2^2/4$, and taking a union bound with the probability of success for Theorem 6.7, we derive Theorem 6.26. $\square$

Part III

Stability and Generalization

Chapter 7

Stability and Generalization of Learning Algorithms

Recipe 10: Cranberry Pie with Streusel Topping

Ingredients

- Pie dough, enough for 1 crust (see Recipe 1)
- 4 $\frac{1}{2}$ cups cranberries
- 1 $\frac{1}{3}$ cups sugar
- Zest of 1 orange
- $\frac{1}{2}$ teaspoon salt
- 1 tablespoon cornstarch
- $\frac{2}{3}$ cup rolled oats
- $\frac{1}{2}$ cup flour
- $\frac{1}{3}$ cup brown sugar
- $\frac{1}{2}$ teaspoon cinnamon
- $\frac{3}{4}$ cup pecans, toasted
- 6 tablespoons butter, melted and cooled

Preparation

1. Preheat oven to 375° F. Roll dough out and transfer to a 9-inch pie pan. Trim overhang and return to fridge.
2. Combine cranberries, 1 cup sugar, orange zest, salt, and cornstarch in a saucepan over medium heat. Cook for 10 minutes, stirring often, until filling is loose. Let cool and add to pie.

3. Grind oats in a food processor. Add pecans and coarsely grind. Add flour, brown sugar, $\frac{1}{3}$ cup sugar, and cinnamon, pulsing to combine. Add butter and pulse until crumbles form. Sprinkle over cranberry filling.
4. Bake for 45 to 50 minutes, until juices are bubbling enough to splash just over the crumb topping.

7.1 Background

Articles, papers, and news stories everywhere abound with references to the success of machine learning. Hundreds of startups tout the power and results of their machine learning systems. While these systems are undeniably powerful and have accomplished tasks that were previously thought near impossible, many of these claims of success suffer from the same existential question that humanity faces: How do we define success?

In some scenarios, success of a machine learning system is easy to define. Consider the setting of *machine diagnosis*, where we want to use machine learning to diagnose incoming patients to a hospital. Ignoring potential privacy issues, we could train a machine learning model on the hospital's patient records. One measure of success could then be "After training, how many past patients did the machine correctly diagnose?"

This measure ignores some important realities. Imagine that we wish to evaluate how well a class of students understood last night's homework by giving them a pop quiz. One option would be to take questions directly from the homework and put them on the quiz. This only tests how well the students understood that specific homework assignment. It does not accurately measure how well they understood the underlying

concepts. To measure this, we would need questions the students had not previously seen. The same is true of machine learning systems. In order to evaluate their success, we need to test them on data that they have not already seen.

This leads to the concepts of *training sets* and *test sets*. We train our machine learning algorithm on the training set and then evaluate it on the test set. In the hospital setting, we could train our algorithm on 90% of the previous patient records, and then test its success on the remaining 10%. The measure of success could then be "After training, how many patients in the test set did the machine correctly diagnose?"

This is not the end of the story. Suppose that Hospital A is located in Madison, Wisconsin. Hospital B, located in San Diego, California, hears about the "success" of the machine learning system in Hospital A, and ask if they can use it to diagnose their patients. While this might ostensibly seem like a good idea, the patients in Madison could be very different than those in San Diego. Geographic, ethnographic, economic, and other factors all contribute to health. The machine learning system in Madison has only ever seen patients from Madison, and we've only ever tested it on patients in Madison. In order to apply it to San Diego patients, we would like some indication that the machine learning model will *generalize* to the patient population in San Diego. This leads to a new measure of success, namely "How well does the trained machine learning system generalize?"

Empirically, the answer seems to be that many modern machine learning algorithms generalize extremely well [105, 57]. This is especially true of models trained using *neural networks*, especially *deep neural networks*. These algorithms often have zero error on their training set and almost zero error on the test set. In the analogy to the homework and quiz for students, modern machine learning systems often get all of the homework

questions correct and almost all of the quiz correct.

Although there has been significant recent work in analyzing the success of such algorithms on their training set, our theoretical understanding of their generalization properties falls far below what has been observed empirically. A useful proxy for analyzing the generalization performance of learning algorithms is that of *stability*. Note that this is a different notion of stability than the one given in Part I of this thesis. An algorithm is stable if small changes in the training set result in small differences in the output predictions of the trained model. A student would be stable if changing their homework slightly did not impact how well they learned the underlying concepts.

In their foundational work, Bousquet and Elisseeff [12] establish that *stability begets generalization*. While generalization is hard to understand, since it often involves quantifying data we have not seen yet, stability can be more tractable. Moreover, the huge variety in machine learning algorithms and the settings they are applied to mean that we would like results that do not just apply to a single algorithm or setting. The goal of this work is to provide easy-to-use stability results that apply to large classes of algorithms and machine learning scenarios.

7.2 Prior work

The idea of stability analysis has been around for more than 30 years since the work of Devroye and Wagner [25]. Bousquet and Elisseeff [12] defined several notions of algorithmic stability and used them to derive bounds on generalization error. Further work has focused on stability of randomized algorithms [32] and the interplay between uniform convergence and generalization [85]. Mukherjee et al. [66] show that stability

implies consistency of empirical risk minimization. Shalev-Shwartz et al. [85] show that stability can also imply learnability in some problems.

While there has been stability analysis for empirical risk minimizers [12, 66], there are far fewer results for commonly used iterative learning algorithms. In a recent novel work, Hardt et al. [43] establish stability bounds for SGD, and discuss algorithmic heuristics that provably increase the stability of SGD models. Unfortunately, generalizing their techniques to establish stability bounds for other first-order methods can be a strenuous task. Showing non-trivial stability for more involved algorithms like SVRG [46], or SGD in more nuanced non-convex setups is far from straightforward. While [43] provides a clean and elegant analysis that shows stability of SGD for non-convex loss functions, the result requires very small step-sizes. The step-size is small enough that one may require exponentially many steps for provable convergence to an approximate critical point, under standard smoothness assumptions [38]. Generally, there seems to be a trade-off between convergence and stability of algorithms.

The work by Lin et al. [58] shows that stability of SGD can be controlled by forms of regularization. In [52], the authors give stability bounds for SGD that are data dependent. Since they do not rely on worst-case arguments, they lead to smaller generalization error bounds than that in [43], but require assumptions on the underlying data. The work by Liu et al. [60] gives a related notion of *uniform hypothesis stability* and show that it implies guarantees on the generalization error.

Stability is closely related to the notion of *differential privacy* introduced in [28]. Roughly speaking, differential privacy ensures that the probability of observing any outcome from a statistical query changes if you modify any single dataset element. Dwork et al. later showed that differentially private algorithms generalize well [29].

These generalization bounds were later improved by Nissim and Stemmer [71]. Such generalization bounds are similar to those guaranteed by stability but often require different tools to handle directly.

7.2.1 Our Contributions

We establish that models trained by algorithms that converge to local minima are stable under the Polyak-Łojasiewicz (PL) and the quadratic growth (QG) conditions [48]. Informally, these conditions assert that the suboptimality of a model is upper bounded by the norm of its gradient and lower bounded by its distance to the closest global minimizer. As we see in the following, these conditions are sufficient for stability and are general enough to yield useful bounds for a variety of settings.

Our results require weaker conditions compared to the state-of-the art, while recovering several prior stability bounds. For example in [43] the authors require convexity, or strong convexity. Gonen and Shalev-Shwartz prove the stability of ERMs for non-convex, but locally strongly convex loss functions obeying strict saddle inequalities [39]. By contrast, we develop comparable stability results for a large class of functions, where no convexity, local convexity, or saddle point conditions are imposed. We note that although [43] establishes the stability of SGD for smooth non-convex objectives, the step-size selection can be prohibitively small for convergence. In our bounds, we make no assumptions on the hyper-parameters of the algorithms.

We use our black-box results to directly compare the generalization performance of popular first-order methods in general learning setups. While direct proofs of stability seem to require a substantial amount of algorithm-specific analysis, our results are

derived from known convergence rates of popular algorithms. For strong convexity—a special case of the PL condition—we recover order-wise the stability bounds of Hardt et al. [43], but for a large family of optimization algorithms (*e.g.,* SGD, GD, SVRG, etc). We show that many of these algorithms offer order-wise similar stability as saddle-point avoiding algorithms in non-convex problems where all local minima are global [39]. We finally show that while SGD and GD have analogous stability in the convex setting, this breaks down in the non-convex setting. We give an explicit example of a simple 1-layer neural network on which SGD is stable but GD is not. Such an example was theorized in [43] (*i.e.,* Figure 10 in the aforementioned paper); here we formalize the authors' intuition. Our results offer yet another indication that SGD trained models can be more generalizable than full-batch GD ones.

Finally, we give examples of some machine learning scenarios where the PL condition mentioned above holds true. Adapting techniques from [42], we show that deep networks with linear activation functions are PL almost everywhere in the parameter space. Our theory allows us to derive results similar to those in [49] about local/global minimizers in linear neural networks.

7.3 Algorithmic Stability

Let $S = \{z_1, \ldots, z_n\}$ be a set of training data, where $z_i \overset{iid}{\sim} \mathcal{D}$. For a model w and a loss function ℓ, let $\ell(w; z)$ be the error of w on the training example z. We define the *expected risk* of a model w by $R[w] := \mathbb{E}_{z\sim\mathcal{D}}\ell(w; z)$. Since, we do not have access to the underlying distribution $\mathcal{D}$ optimizing $R[w]$ directly is not possible. Instead, we will

measure the *empirical risk* of a model w on a set S, given by:

$$R_S[w] := \frac{1}{n}\sum_{i=1}^{n} \ell(w; z_i).$$

The generalization performance of the model can then be measured by the *generalization gap*:

$$\epsilon_{\text{gen}}(w) := |R_S[w] - R[w]|.$$

For our purposes, w will be the output of some (potentially randomized) learning algorithm $\mathcal{A}$, trained on some data set S. We will denote this output by $\mathcal{A}(S)$.

Let us now define a related training set $S' = \{z_1, \ldots, z_{i-1}, z_i', z_{i+1}, \ldots, z_n\}$, where $z_i' \sim \mathcal{D}$. We then have the following notion of uniform stability that was first introduced in [12].

Definition 7.1 (Uniform Stability)**.** *An algorithm $\mathcal{A}$ is uniformly ϵ-stable, if for all data sets S, S' differing in at most one example, we have*

$$\sup_z \mathbb{E}_{\mathcal{A}}\big[\ell(\mathcal{A}(S); z) - \ell(\mathcal{A}(S'); z)\big] \leq \epsilon.$$

The expectation is taken with respect to the (potential) randomness of the algorithm $\mathcal{A}$. Bousquet and Elisseeff establish that uniform stability implies small generalization gap [12].

Theorem 7.2. *Suppose $\mathcal{A}$ is uniformly ϵ-stable. Then,*

$$|\mathbb{E}_{S,\mathcal{A}}\big[R_S[\mathcal{A}(S)] - R[\mathcal{A}(S)]\big]| \leq \epsilon.$$

In practice, uniform stability may be too restrictive, since the bound above must hold for all z, irrespective of its marginal distribution. The following notion of stability, while weaker, is still enough to control the generalization gap. Given a data set $S = \{z_1, \ldots, z_n\}$ and $i \in \{1, \ldots, n\}$, we define S^i as $S \backslash z_i$.

Definition 7.3 (Pointwise Hypothesis Stability, [12]). *$\mathcal{A}$ has pointwise hypothesis stability β with respect to a loss function ℓ if*

$$\forall i \in \{1, \ldots, n\},\ \mathbb{E}_{\mathcal{A},S}\big[|\ell(\mathcal{A}(S); z_i) - \ell(\mathcal{A}(S^i); z_i)|\big] \leq \beta.$$

Note that this is a weaker notion than uniform stability, but one can still use it to establish non-trivial generalization bounds:

Theorem 7.4 ([12]). *Suppose we have a learning algorithm $\mathcal{A}$ with pointwise hypothesis stability ϵ with respect to a bounded loss function ℓ such that $0 \leq \ell(w; z) \leq M$. For any δ, we have with probability at least $1 - \delta$,*

$$R[\mathcal{A}(S)] \leq R_S[\mathcal{A}(S)] + \sqrt{\frac{M^2 + 12Mn\epsilon}{2n\delta}}.$$

7.3.1 Stability and (Strongly) Convex Loss Functions

As above, we suppose that we have a training set S and a loss function $\ell(w; z)$ that measures the accuracy of the model w on the training example z. As it turns out, the geometry of ℓ is an important factor in the stability of algorithms that try to minimize the empirical risk function, $R_S[w]$. In particular, prior work has shown that strongly convex functions have desirable stability properties.

Recall that $f : \mathbb{R}^n \to \mathbb{R}$ is strongly convex with parameter λ if for all $x, y \in \mathbb{R}^n$,

$$f(y) \geq f(x) + \langle \nabla f(x), y - x \rangle + \frac{\lambda}{2}\|y - x\|_2^2.$$

We then have the following theorem, a corollary of work due to Bousquet and Elisseeff [12], about empirical risk minimizers of strongly convex functions.

Theorem 7.5. *Suppose that for all z, $\ell(\cdot, z)$ is λ-strongly convex and L-Lipschitz. For any training set S of size n, let $\mathcal{A}(S)$ be the empirical risk minimizer of $R_S[w]$. In other words, $\mathcal{A}$ computes*

$$\mathcal{A}(S) = \operatorname*{argmin}_w \frac{1}{n} \sum_{z \in S} \ell(w; z).$$

Then $\mathcal{A}$ is uniformly stable with stability ϵ_{stab} where

$$\epsilon_{stab} \leq \frac{L^2}{\lambda n}.$$

In practice, it is difficult to compute the exact empirical risk minimizer, even when we have strongly convex loss functions. Instead, we are primarily interested in the stability of frequently used algorithms, such as SGD and gradient descent. To this end, work in [43] bounds the stability of SGD in such scenarios. They show the following theorem.

Theorem 7.6. *Suppose that $\ell(\cdot, z)$ is λ-strongly convex, β-smooth, and L-Lipschitz for all z. If we run SGD with a constant step-size $\gamma \leq 1/\beta$ for T iterations, then SGD will be uniformly stable with stability ϵ_{stab} where*

$$\epsilon_{stab} \leq \frac{2L^2}{\lambda n}.$$

This is the same stability as that of the empirical risk minimizer, up to a factor of 2. Even though we do not necessarily converge to the exact empirical risk minimizer, the geometry of the loss function still allows us to derive useful stability results.

7.4 Mathematical Perspective and Main Results

As discussed above, the geometry of strongly convex functions allows us to derive stability results for empirical risk minimizers and SGD applied to strongly convex functions.

While [43] also derive results for SGD in the non-convex setting, they require the step-size at iteration t to satisfy $\gamma_t \leq c/t$ for some constant c. This may result in exponentially slow convergence.

The other issue is that the analysis in [43] does not directly generalize to similar algorithms. We would like to derive stability results that apply to non-convex functions and large classes of algorithms. Simply put, an algorithm-by-algorithm stability analysis would be a tedious, drawn-out process. Moreover, varying assumptions in theory may make comparing stability of algorithms difficult.

Instead, our main results will show that we can decompose the stability ϵ_{stab} as

$$\epsilon_{stab} \leq \epsilon_{\mathcal{A}} + \delta$$

where $\epsilon_{\mathcal{A}}$ only relies on the convergence of the algorithm $\mathcal{A}$, and δ is a function of the geometry of the underlying loss function. Moreover, to do this we do not need strong convexity, or even convexity. To get δ comparable to previous stability results for strongly convex functions, we only need our loss functions to satisfy the Polyak-Łojasiewicz (PL) condition. A function $f : \mathbb{R}^n \to \mathbb{R}$ is said to satisfy this condition with parameter $\mu > 0$ if for all $x \in \mathbb{R}^n$,

$$\frac{1}{2}\|\nabla f(x)\|_2^2 \geq \mu(f(x) - f^*).$$

As we discuss in the following, the PL condition allows for non-convex functions, but constrains the geometry in a manner similar to strong convexity. This allows us to derive parallel results between strongly convex and PL functions, such as linear convergence of gradient descent. As we show, it also allows us to derive similar stability results between strongly convex and PL functions, for large classes of algorithms. We will show the following theorem.

Theorem 7.7. *Suppose that we use an algorithm $\mathcal{A}$ to minimize the empirical loss function $R_S[w]$ for a training set of size n. If $R_S[w]$ is PL with parameter μ, then $\mathcal{A}$ will be pointwise hypothesis stable with stability ϵ_{stab} satisfying*

$$\epsilon_{stab} \leq \epsilon_{\mathcal{A}} + \frac{2L^2}{\mu(n-1)}$$

where $\epsilon_{\mathcal{A}}$ depends only on the convergence of $\mathcal{A}$.

We will also show a version of this theorem when $R_S[w]$ only satisfies a weaker condition called Quadratic Growth. This will be discussed in the following. We then use these results to understand the stability of popular first-order algorithms.

Despite this algorithm-agnostic result, we will show that when we consider more general non-convex loss functions, different algorithms can exhibit wildly different stability. In particular, we construct an explicit non-convex loss function and training sets where gradient descent is not stable but SGD is.

Chapter 8

Stability and the Polyak-Łojasiewicz Condition

Recipe 11: Blackberry Cheesecake Galette

Ingredients

- Pie dough, enough for 1 crust (see Recipe 1)
- 1 cup blackberries, halved
- $\frac{1}{2}$ cup sugar
- 1 teaspoon lime juice
- 1 teaspoon cornstarch
- 8 ounces cream cheese, softened
- 1 egg and 1 egg, separated
- Zest of $\frac{1}{2}$ lime
- $\frac{1}{2}$ teaspoon vanilla extract
- $\frac{1}{4}$ teaspoon salt
- 1 tablespoon chopped pistachios

Preparation

1. In a small bowl, combine blackberries, 1 tablespoon sugar, lime juice, and cornstarch. Stir and set aside.
2. Preheat oven to 350° F. Roll dough out into large 12-14 inch circle and transfer to a 9-inch pie pan. Do not trim overhang.
3. In a medium bowl, beat cream cheese with whole egg and egg white until

light and fluffy. Beat in remaining sugar, lime zest, vanilla, and salt.

4. Pour mixture into prepared pie dish. Spoon blackberry mixture and juices over the cream cheese mixture. Swirl lightly with a toothpick for decoration.

5. Gently lift the overhanging dough and pinch into loose creases. Gently lay crease down over filling. Repeat until no overhang remains.

6. Combine egg yolk and $\frac{1}{2}$ teaspoon water. Gently brush crust with egg mixture. Sprinkle the entire tart with sugar and pistachios.

7. Bake for 35 minutes or until a tester inserted into the cheesecake comes out clean.

8.1 The Polyak-Łojasiewicz and Quadratic Growth Conditions

In the following, we derive stability for models trained on empirical risk functions satisfying the Polyak-Łojasiewicz (PL) and the Quadratic Growth (QG) condition. We assume throughout that the functions in question are L-Lipschitz. For simplicity of notation, we will assume that all domains are subsets of $\mathbb{R}^m$ for some m. We will typically consider the ℓ_2 norm on such domains.

In [48], Karimi et al. used the *Polyak-Łojasiewicz condition* to prove simplified nearly optimal convergence rates for several first-order methods. Notably, there are some non-convex functions that satisfy the PL condition. The condition is defined below.

Definition 8.1 (Polyak-Łojasiewicz)**.** *Fix a set $\mathcal{X}$ and let f^* denote the minimum value*

of f on $\mathcal{X}$. We will say that a function f satisfies the Polyak-Łojasiewicz (PL) condition on $\mathcal{X}$, if there exists $\mu > 0$ such that for all $x \in \mathcal{X}$ we have

$$\frac{1}{2}\|\nabla f(x)\|_2^2 \geq \mu(f(x) - f^*). \tag{8.1}$$

This condition roughly says that the gradient of f grows quadratically in norm as we move away from a global minimizer. This geometric condition is similar to strong convexity, which bounds the function from below by a quadratic function. Note that for PL functions, every critical point is a global minimizer. This is a weaker condition than being strongly-convex, as the following lemma shows.

Lemma 8.2. *Suppose that f is λ-strongly convex. Then f is λ-PL.*

Proof. Since f is λ-strongly convex, we know that for all x, y in the domain of f, we have

$$f(y) \geq f(x) + \langle \nabla f(x), y - x \rangle + \frac{\lambda}{2}\|y - x\|_2^2.$$

Setting y to be the unique minimizer x^* of $f(x)$, we then get

$$\begin{aligned} f(x^*) &\geq f(x) + \langle \nabla f(x), x^* - x \rangle + \frac{\lambda}{2}\|x^* - x\|_2^2 \\ &\geq f(x) + \min_z \langle \nabla f(x), z - x \rangle + \frac{\lambda}{2}\|z - x\|_2^2. \end{aligned}$$

Define

$$g(z) := \langle \nabla f(x), z - x \rangle + \frac{\lambda}{2}\|z - x\|_2^2.$$

Setting the gradient equal to zero, we get

$$\begin{aligned} \nabla g(z) &= \nabla f(x) + \lambda(z - x) = 0. \\ \implies z &= x - \frac{1}{\lambda}\nabla f(x). \end{aligned}$$

At this point, we then have

$$g\left(x - \frac{1}{\lambda}\nabla f(x)\right) = -\frac{1}{2\lambda}\|\nabla f(x)\|_2^2.$$

Therefore,

$$\begin{aligned} f(x^*) &\geq f(x) + \min_z g(z) \\ &= f(x) - \frac{1}{2\lambda}\|\nabla f(x)\|_2^2. \end{aligned}$$

Rearranging, we find

$$\frac{1}{2}\|\nabla f(x)\|_2^2 \geq \lambda(f(x) - f^*).$$

□

In general, the converse does not hold. There are functions that are PL but not strongly convex.

Example 8.3. *Consider the function $f(x, y) = x^2$. Then note that f does not have a unique minimizer. Its minimizers are all points of the form $(0, y)$ and these points all satisfy $f(0, y) = 0$. However, we have*

$$\begin{aligned} \frac{1}{2}\|\nabla f(x, y)\|_2^2 &= \frac{1}{2}\left\|[2x \quad 0]^T\right\|_2^2 \\ &= 2x^2 \\ &\geq 2(x^2) \\ &= 2(f(x)) \\ &= 2(f(x) - f^*). \end{aligned}$$

Therefore, f is not strongly convex but it is PL with $\mu = 2$.

In general, PL functions are examples of invex functions.

Definition 8.4 (Invex)**.** *A differentiable function $f : \mathbb{R}^n \to \mathbb{R}$ is invex if every critical point is a global minimizer.*

Clearly, PL functions satisfy this property. We also have the following equivalent definition of the PL condition due to Karimi et al.

Lemma 8.5 (Error bound, [48])**.** *Let $f : \mathbb{R}^n \to \mathbb{R}$. For any $x \in \mathbb{R}^n$, let x_p denote the closest global optima in $\mathcal{X}$. Then f satisfies the PL condition iff there is some constant $\mu > 0$ such that for all $x \in \mathbb{R}^n$,*

$$\|\nabla f(x)\|_2 \geq \mu \|x_p - x\|_2.$$

PL functions are important in optimization due to the fact that under this condition, many first-order methods exhibit similar convergence rates to the strongly convex case. We give one theorem below showing that under the PL condition, gradient descent has a linear convergence rate, just as in the strongly convex case. The theorem and proof were first shown in [48]. We also present the proof to help make the reader familiar with how one can use the PL condition in practice to derive practical results.

Theorem 8.6 ([48])**.** *Suppose that $f(x)$ is β-smooth and satisfies the PL condition with parameter μ. Then gradient descent on f with step-size $1/\beta$ with updates*

$$x_{k+1} = x_k - \frac{1}{\beta} \nabla f(x_k)$$

has a global linear convergence rate

$$f(x_k) - f^* \leq \left(1 - \frac{\mu}{\beta}\right)^k (f(x_0) - f^*).$$

Proof. Setting $y = x_{k+1}, x = x_k$ in (2.2), we have

$$\begin{aligned} f(x_{k+1}) - f(x_k) &\leq \langle \nabla f(x_k), -\frac{1}{\beta}\nabla f(x_k)\rangle + \frac{\beta}{2}\|\frac{1}{\beta}\nabla f(x_k)\|_2^2 \\ &= -\frac{1}{2\beta}\|\nabla f(x_k)\|_2^2. \end{aligned}$$

By the PL inequality in (8.1), this implies

$$f(x_{k+1}) - f(x_k) \leq -\frac{\mu}{\beta}(f(x_k) - f^*).$$

Rearranging, this implies

$$\begin{aligned} f(x_{k+1}) - f^* &\leq \left(1 - \frac{\mu}{\beta}\right)(f(x_k) - f^*) \\ &\leq \left(1 - \frac{\mu}{\beta}\right)^2 (f(x_{k-1}) - f^*) \\ &\vdots \\ &\leq \left(1 - \frac{\mu}{\beta}\right)^{k+1} (f(x_0) - f^*). \end{aligned}$$

□

[48] also shows that under the PL condition, many first-order algorithms, including SGD, RCD, Proximal-Gradient Descent, and SVRG, exhibit similar convergence rates to the setting where the function is strongly convex. In this paper, we also consider a strictly larger family of functions that satisfy the Quadratic Growth (QG) condition.

Definition 8.7 (Quadratic Growth)**.** *We will say that a function f satisfies the quadratic growth (QG) condition on a set $\mathcal{X}$, if there exists $\mu > 0$ such that for all $x \in \mathcal{X}$ we have*

$$f(x) - f^* \geq \frac{\mu}{2}\|x_p - x^*\|_2^2,$$

where x_p denotes the Euclidean projection of x onto the set of global minimizers of f in $\mathcal{X}$ (i.e., x_p is the closest point to x in $\mathcal{X}$ satisfying $f(x_p) = f^*$).

Note that this is weaker than the PL condition, as shown in [48].

Lemma 8.8 ([48])**.** *The PL condition implies the QG condition.*

Both of these conditions have been considered in previous studies. The PL condition was first introduced by Polyak in [61], who showed that under this assumption, gradient descent converges linearly. The QG condition has been considered under various guises [10, 45] and can imply important properties about the geometry of critical points. For example, [2] showed that local minima of nonlinear programs satisfying the QG condition are actually isolated stationary points. These kinds of geometric implications will allow us to derive stability results for large classes of algorithms.

8.2 Black-box Stability of Approximate Global Minima

In this section, we establish the stability of large classes of learning algorithms under the PL and QG conditions presented above. Our stability results are "black-box" in the sense that our bounds are decomposed as a sum of two terms: a term concerning the convergence of the algorithm to a global minimizer, and a term relevant to the geometry of the loss function around the global minima. Both terms are used to establish good generalization and provide some insights into the way that learning algorithms perform.

For a given data set S, suppose we use an algorithm $\mathcal{A}$ to train some model w. We let w_S denote the output of our algorithm on S. The empirical training error on a data set S is denoted $f_S(w)$ and is given by

$$f_S(w) = \frac{1}{|S|} \sum_{z \in S} \ell(w; z).$$

We assume that each of these losses is L-Lipschitz with respect to the parameters of the model. We are interested in conditions on f_S that allow us to make guarantees on the stability of $\mathcal{A}$. As it turns out, the PL and QG condition will allow us to prove such results. Although it seems unclear if these conditions are reasonable, in our last section we show that they arise in a large number of machine learning settings, including in certain deep neural networks.

8.2.1 Pointwise Hypothesis Stability for PL/QG Loss Functions

To analyze the performance machine learning algorithms, it often suffices to understand the algorithm's behavior with respect to critical points. This requires knowledge of the convergence of the algorithm, and an understanding of the geometric properties of the loss function around critical points. As it turns out, the PL and QG conditions allow us to understand the geometry underlying the minima of our function. Let $\mathcal{X}_{\min}$ denote the set of global minima of f_S.

Theorem 8.9. *Assume that for all S and $w \in \mathcal{X}$, f_S is PL with parameter μ. We assume that applying $\mathcal{A}$ to f_S produces output w_S that is converging to some global minimizer w_S^*. Then $\mathcal{A}$ has pointwise hypothesis stability with parameter ϵ_{stab} satisfying the following conditions.*

Case 1: *If for all S, $\|w_S - w_S^*\|_2 \leq O(\epsilon_{\mathcal{A}})$ then*

$$\epsilon_{stab} \leq O(L\epsilon_{\mathcal{A}}) + \frac{2L^2}{\mu(n-1)}.$$

Case 2: *If for all S, $|f_S(w_S) - f_S(w_S^*)| \leq O(\epsilon'_{\mathcal{A}})$ then*

$$\epsilon_{stab} \leq O\left(L\sqrt{\frac{\epsilon'_{\mathcal{A}}}{\mu}}\right) + \frac{2L^2}{\mu(n-1)}.$$

Case 3: *If for all S, $\|\nabla f_S(w_S)\|_2 \leq O(\epsilon''_{\mathcal{A}})$, then*

$$\epsilon_{stab} \leq O\left(\frac{L\epsilon''_{\mathcal{A}}}{\mu}\right) + \frac{2L^2}{\mu(n-1)}.$$

Proof. Fix a training set S and $i \in \{1, \ldots, n\}$. We will show pointwise hypothesis stability for all S, i instead of for them in expectation. Let w_1 denote the output of $\mathcal{A}$ on S, and let w_2 denote the output of $\mathcal{A}$ on S^i. Let w_1^* denote the critical point of f_S to which w_1 is approaching, and w_2^* denote the critical point of f_{S^i} that w_2 is approaching. We then have,

$$\begin{aligned}
&|\ell(w_1; z_i) - \ell(w_2; z_i)| \\
&\leq |\ell(w_1; z_i) - \ell(w_1^*; z_i)| + |\ell(w_1^*; z_i) - \ell(w_2^*; z_i)| + |\ell(w_2^*; z_i) - \ell(w_2; z_i)|. \qquad (8.2)
\end{aligned}$$

We first wish to bound the first and third terms of (8.2). The bound depends on the case in Theorem 8.9.

Case 1: By assumption, $\|w_1 - w_1^*\|_2 = O(\epsilon_{\mathcal{A}})$. Since $\ell(\cdot; z_i)$ is L-Lipschitz, this implies

$$|\ell(w_1; z_i) - \ell(w_1^*; z_i)| \leq L\|w_1 - w_1^*\|_2 = O(L\epsilon_{\mathcal{A}}).$$

Case 2: As stated in Lemma 8.8, the PL condition implies the QG condition. Therefore,

$$\frac{\mu}{2}\|w_1 - w_1^*\|_2^2 \leq |f_S(w_1) - f_S(w_1^*)|. \qquad (8.3)$$

By assumption on case 2, $|f_S(w_1) - f_S(w_1^*)| \leq O(\epsilon'_{\mathcal{A}})$. This implies

$$\begin{aligned}\|w_1 - w_1^*\|_2 &\leq \frac{\sqrt{2}}{\sqrt{\mu}}\sqrt{|f_S(w_1) - f_S(w_1^*)|} \\ &= O\left(\sqrt{\frac{\epsilon'_{\mathcal{A}}}{\mu}}\right).\end{aligned}$$

Case 3: By Lemma 8.5, the PL condition on w_1, w_1^* implies that

$$\|\nabla f_S(w_1)\|_2 \geq \mu\|w_1 - w_1^*\|_2.$$

Using the fact that f_S is L-Lipschitz and the fact that $\|\nabla f_S(w_j)\|_2 \leq O(\epsilon''_{\mathcal{A}})$ by assumption on Case 3, we find

$$|\ell(w_1; z_i) - \ell(w_1^*; z_i)| \leq L\|w_1 - w_1^*\|_2 \leq \frac{L}{\mu}\|\nabla f_S(w_1)\|_2 \leq O\left(\frac{L\epsilon''_{\mathcal{A}}}{\mu}\right).$$

In the above three cases, we can bound $|\ell(w_2; z_i) - \ell(w_2^*; z_i)|$ in the same manner. We now wish to bound the second term of (8.2). Note that we can manipulate this term as

$$\begin{aligned}|\ell(w_1^*; z_i) - \ell(w_2^*; z_i)| &= |(nf_S(w_1^*) - (n-1)f_{S^i}(w_1^*)) - (nf_S(w_2^*) + (n-1)f_{S^i}(w_2^*))| \\ &\leq n|f_S(w_1^*) - f_S(w_2^*)| + (n-1)|f_{S^i}(w_1^*) - f_{S^i}(w_2^*)|. \qquad (8.4)\end{aligned}$$

By the PL condition, we can find a local minima u of f_S such that

$$\|\nabla f_S(w_2^*)\|_2^2 \geq \mu|f_S(w_2^*) - f_S(u)|.$$

Similarly, we can find a local minima v of f_{S^i} such that

$$\|\nabla f_{S^i}(w_1^*)\|_2^2 \geq \mu|f_{S^i}(w_1^*) - f_{S^i}(v)|.$$

Note that since $\nabla f_{S^i}(w_2^*) = 0$, we get:

$$\|\nabla f_S(w_2^*)\|_2^2 = \frac{1}{n^2}\|\nabla \ell(w_2^*; z_i)\|_2^2 \leq \frac{L^2}{n^2}.$$

Similarly, since $\nabla f_S(w_1^*) = 0$, we get:

$$\|\nabla f_{S^i}(w_1^*)\|_2^2 = \frac{1}{(n-1)^2}\|\nabla \ell(w_1^*; z_i)\|_2^2 \leq \frac{L^2}{(n-1)^2}.$$

Since all local minima of a PL function are global minima, we obtain

$$\begin{aligned} n|f_S(w_1^*) - f_S(w_2^*)| &\leq n|f_S(w_1^*) - f_S(u)| + n|f_S(u) - f_S(w_2^*)| \\ &\leq n\frac{L^2}{\mu n^2} \\ &= \frac{L^2}{\mu n}. \end{aligned} \tag{8.5}$$

In a similar manner, we get

$$\begin{aligned} (n-1)|f_{S^i}(w_1^*) - f_{S^i}(w_2^*)| &\leq (n-1)|f_{S^i}(w_1^*) - f_{S^i}(v)| + (n-1)|f_{S^i}(v) - f_{S^i}(w_2^*)| \\ &\leq (n-1)\frac{L^2}{\mu(n-1)^2} \\ &\leq \frac{L^2}{\mu(n-1)}. \end{aligned} \tag{8.6}$$

Plugging in 8.5, 8.6 into 8.4, we find

$$|\ell(w_1^*; z_i) - \ell(w_2^*; z_i)| \leq \frac{L^2}{\mu n} + \frac{L^2}{\mu(n-1)}.$$

This proves the desired result.

□

Suppose our loss functions are PL and our algorithm $\mathcal{A}$ is an oracle that returns a global optimizer w_S^*. Then the terms $\epsilon_{\mathcal{A}}, \epsilon'_{\mathcal{A}}, \epsilon''_{\mathcal{A}}$ above are all identical to 0, leading to the following corollary.

Corollary 8.10. *Let f_S satisfy the PL inequality with parameter μ and let*

$$\mathcal{A}(S) = \operatorname*{argmin}_{w \in \mathcal{X}} f_S(w).$$

Then $\mathcal{A}$ has pointwise hypothesis stability with

$$\epsilon_{stab} = \frac{2L^2}{\mu(n-1)}.$$

Bousquet and Elisseeff considered the stability of empirical risk minimizers where the loss function satisfied strong convexity [12]. Their work implies that for λ-strongly convex functions, the empirical risk minimizer has we stability satisfying $\epsilon_{stab} \leq \frac{L^2}{\lambda n}$. Since λ-strongly convex implies λ-PL, Corollary 8.14 generalizes their result, with only a constant factor loss.

Remark 8.11. *Theorem 8.9 holds even if we only have information about $\mathcal{A}$ in expectation. For example, if we only know that $\mathbb{E}_{\mathcal{A}}\|w_S - w_S^*\|_2 \leq O(\epsilon_{\mathcal{A}})$, we still establish pointwise hypothesis stability (in expectation with respect to $\mathcal{A}$), with the same constant as above. This allows us to apply our result to algorithms such as SGD where we are interested in the convergence in expectation.*

A similar result to Theorem 8.9 can be derived for empirical risk functions satisfy the QG condition and are *realizable*, *e.g.,* where zero training loss is achievable.

Theorem 8.12. *Assume that for all S and $w \in \mathcal{X}$, f_S is QG with parameter μ and that all its global minima u satisfy $f_S(u) = 0$. We assume that applying $\mathcal{A}$ to f_S produces output w_S that is converging to some global minimizer w_S^*. We also assume that for all feasible w and z, $|\ell(w; z)| \leq c$. Then $\mathcal{A}$ has pointwise hypothesis stability with parameter ϵ_{stab} satisfying the following conditions.*

Case 1: *If for all S, $\|w_S - w_S^*\|_2 \leq O(\epsilon_{\mathcal{A}})$ then*

$$\epsilon_{stab} \leq O(L\epsilon_{\mathcal{A}}) + 2L\sqrt{\frac{c}{\mu n}}.$$

Case 2: *If for all S, $|f_S(w_S) - f_S(w_S^*)| \leq O(\epsilon'_{\mathcal{A}})$ then*

$$\epsilon_{stab} \leq O\left(L\sqrt{\frac{\epsilon'_{\mathcal{A}}}{\mu}}\right) + 2L\sqrt{\frac{c}{\mu n}}.$$

Proof. Fix a training set S and $i \in \{1, \ldots, n\}$. We will show pointwise hypothesis stability for all S, i instead of for them in expectation. Let w_1 denote the output of $\mathcal{A}$ on S, and let w_2 denote the output of $\mathcal{A}$ on S^i. Let w_1^* denote the critical point of f_S to which w_1 is approaching, and w_2^* denote the critical point of f_{S^i} that w_2 is approaching. We then have,

$$\begin{aligned}
&|\ell(w_1; z_i) - \ell(w_2; z_i)| \\
&\leq |\ell(w_1; z_i) - \ell(w_1^*; z_i)| + |\ell(w_1^*; z_i) - \ell(w_2^*; z_i)| + |\ell(w_2^*; z_i) - \ell(w_2; z_i)|. \qquad (8.7)
\end{aligned}$$

We first wish to bound the first and third terms of (8.7). The bound depends on the case in Theorem 8.12.

Case 1: By assumption, $\|w_1 - w_1^*\|_2 = O(\epsilon_{\mathcal{A}})$. Since $\ell(\cdot; z_i)$ is L-Lipschitz, this implies

$$|\ell(w_1; z_i) - \ell(w_1^*; z_i)| \leq L\|w_1 - w_1^*\|_2 = O(L\epsilon_{\mathcal{A}}).$$

Case 2: By the QG condition, we have

$$\frac{\mu}{2}\|w_1 - w_1^*\|_2^2 \leq |f_S(w_1) - f_S(w_1^*)|. \qquad (8.8)$$

By assumption on case 2, $|f_S(w_1) - f_S(w_1^*)| \leq O(\epsilon'_{\mathcal{A}})$. This implies

$$\begin{aligned}\|w_1 - w_1^*\|_2 &\leq \frac{\sqrt{2}}{\sqrt{\mu}}\sqrt{|f_S(w_1) - f_S(w_1^*)|} \\ &= O\left(\sqrt{\frac{\epsilon'_{\mathcal{A}}}{\mu}}\right).\end{aligned}$$

In the above two cases, we can bound $|\ell(w_2; z_i) - \ell(w_2^*; z_i)|$ in the same manner. We now wish to bound the second term of (8.7). By the QG property, we can pick some local minima v of f_S such that

$$\|w_2^* - v\|_2 \leq \frac{2}{\sqrt{\mu}}\sqrt{|f_S(w_2^*) - f_S(v)|}. \tag{8.9}$$

We then have

$$|\ell(w_1^*; z_i) - \ell(w_2^*; z_i)| \leq |\ell(w_1^*; z_i) - \ell(v; z_i)| + |\ell(v; z_i) - \ell(w_2^*; z_i)|.$$

Note that by assumption, $f_S(w_1^*) = f_S(u) = 0$, so $|\ell(w_1^*; z_i) - \ell(v; z_i)| = 0$. By the Lipschitz property and the QG condition we have

$$\begin{aligned}|\ell(v; z_i) - \ell(w_2^*; z_i)| &\leq L\|v - w_2^*\|_2 \\ &\leq \frac{2L}{\sqrt{\mu}}\sqrt{|f_S(w_2^*) - f_S(v)|} \\ &\leq \frac{2L}{\sqrt{\mu}}\sqrt{|f_S(w_2^*) - f_S(w_1^*)| + |f_S(w_1^*) - f_S(v)|}. \end{aligned}\tag{8.10}$$

Note that $|f_S(w_1^*) - f_S(v)| = 0$ by our realizability assumption. By assumption on w_1^*, w_2^*, we know that $f_S(w_1^*) \leq f_S(w_2^*)$ and $f_{S^i}(w_2^*) \leq f_{S^i}(w_1^*)$. Some simple analysis shows

$$\begin{aligned}nf_S(w_2^*) &= nf_{S^i}(w_2^*) + \ell(w_2^*; z_i) \\ &\leq nf_{S^i}(w_1^*) + \ell(w_2^*; z_i) \\ &= nf_S(w_1^*) + \ell(w_2^*; z_i) - \ell(w_1^*; z_i).\end{aligned}$$

Since $\ell(w_2^*; z_i) \leq c$, this implies that $n|f_S(w_2^*) - f_S(w_1^*)| \leq c$. Plugging this bound into 8.10, we get

$$|\ell(v; z_i) - \ell(w_2^*; z_i)| \leq 2L\sqrt{\frac{c}{\mu n}}.$$

This proves the desired result. □

Remark 8.13. *Observe that unlike the case of PL empirical losses, QG empirical losses only allow for a $O(\frac{1}{\sqrt{n}})$ convergence rate of stability. Moreover, similarly to our result for PL loss functions, the result of Theorem 8.12 holds even if we only have information about the convergence of $\mathcal{A}$ in expectation.*

Finally, we can obtain the following corollary for empirical risk minimizers of QG loss functions.

Corollary 8.14. *Let f_S satisfy the QG inequality with parameter μ and let*

$$\mathcal{A}(S) = \operatorname*{argmin}_{w \in \mathcal{X}} f_S(w).$$

Then $\mathcal{A}$ has pointwise hypothesis stability with

$$\epsilon_{stab} = 2L\sqrt{\frac{c}{\mu n}}.$$

8.2.2 Uniform Stability for PL/QG Loss Functions

Under a more restrictive setup, we can obtain similar bounds for uniform hypothesis stability, which is a stronger stability notion compare to its pointwise hypothesis variant. The usefulness of uniform stability compared to pointwise stability, is that it can lead to generalization bounds that concentrate exponentially faster [12] with respect to the sample size n.

As before, given a data set S, we let denote w_S be the model that $\mathcal{A}$ outputs. Let $\pi_S(w)$ denote the closest optimal point of f_S to w. We will denote $\pi_S(w_S)$ by w_S^*. Let S, S' be data sets differing in at most one entry. We will make the following technical assumption:

Assumption 8.15. *The empirical risk minimizers for f_S and $f_{S'}$, i.e., $w_S^*, w_{S'}^*$ satisfy $\pi_S(w_{S'}^*) = w_S^*$, where $\pi_S(w)$ is the projection of w on the set of empirical risk minimizers of f_S. Note that this is satisfied if for every data set S, there is a unique minimizer w_S^*.*

Remark 8.16. *We would like to note that the above assumption is extremely strict, and in general does not apply to empirical losses with infinitely many global minima. To tackle the existence of infinitely many global minima, one could imagine designing $\mathcal{A}(S)$ to output a structured empirical risk minimizer, e.g., one such that if $\mathcal{A}$ is applied on S', its projection on the optima of f_S would always yield back $A(S)$. This could be possible, if $A(S)$ corresponded to minimizing instead a regularized, or structure constrained cost function whose set of optimizers only contained a small subset of the global minima of f_S. Unfortunately, coming up with such a structured empirical risk minimizer for general non-convex losses seems far from straightforward, and serves as an interesting open problem.*

Theorem 8.17. *Assume that for all S, f_S satisfies the PL condition with constant μ, and suppose that Assumption 8.15 holds. Then $\mathcal{A}$ has uniform stability with parameter ϵ_{stab} satisfying the following conditions.*

Case 1: *If for all S, $\|w_S - w_S^*\|_2 \leq O(\epsilon_{\mathcal{A}})$ then*

$$\epsilon_{stab} \leq O(L\epsilon_{\mathcal{A}}) + \frac{2L^2}{\mu n}.$$

Case 2: *If for all S, $|f_S(w_S) - f_S(w_S^*)| \leq O(\epsilon'_{\mathcal{A}})$ then*

$$\epsilon_{stab} \leq O\left(L\sqrt{\frac{\epsilon'_{\mathcal{A}}}{\mu}}\right) + \frac{2L^2}{\mu n}.$$

Case 3: *If for all S, $\|\nabla f_S(w_S)\|_2 \leq O(\epsilon''_{\mathcal{A}} s)$, then*

$$\epsilon_{stab} \leq O\left(\frac{L\epsilon''_{\mathcal{A}}}{\mu}\right) + \frac{2L^2}{\mu n}.$$

Proof. Let $S_1 = \{z_1, \ldots, z_n\}, S_2 = \{z_1, \ldots, z_{n-1}, z'_n\}$ be data sets of size n differing only in one entry. Let w_i denote the output of $\mathcal{A}$ on data set S_i and let w_i^* denote $w_{S_i}^*$. Let $f_i(w) = f_{S_i}(w)$.

Using the fact that $\ell(\cdot; z)$ is L-Lipschitz we get

$$\begin{aligned}|\ell(w_1; z) - \ell(w_2; z)| &\leq L\|w_1 - w_2\|_2 \\ &\leq L\|w_1 - w_1^*\|_2 + L\|w_1^* - w_2^*\|_2 + L\|w_2^* - w_2\|_2 \end{aligned} \tag{8.11}$$

Note that by **A1**, we know that w_1^* is the closest optimal point of f_1 to w_2^*. By the PL condition,

$$\begin{aligned}\|w_1^* - w_2^*\|_2 &\leq \frac{1}{\mu}\|\nabla f_1(w_2^*)\|_2 \\ &= \frac{1}{\mu}\|\nabla f_2(w_2^*) - \frac{1}{n}\nabla\ell(w_2^*; z'_n) + \frac{1}{n}\nabla\ell(w_2^*; z_n)\|_2 \\ &\leq \frac{1}{\mu n}(\|\nabla\ell(w_2^*; z'_n)\|_2 + \|\nabla\ell(w_2^*; z_n)\|_2) \\ &\leq \frac{2L}{\mu n}\end{aligned}$$

This bounds the second term of 8.11. The first and third terms must be bounded differently depending on the case.

Case 1: By assumption, for $i = 1, 2$, $\|w_i - w_i^*\|_2 = O(\epsilon_{\mathcal{A}})$, proving the result.

Case 2: As stated in Lemma 8.8, PL implies QG. Therefore for $i = 1, 2$,

$$\begin{aligned}\|w_i - w_i^*\|_2 &\leq \frac{\sqrt{2}}{\sqrt{\mu}}\sqrt{f_i(w_i) - f_i^*} \\ &= O\left(\sqrt{\frac{\epsilon'_{\mathcal{A}}}{\mu}}\right)\end{aligned}$$

Case 3: As mentioned above in Lemma 8.5, the PL condition implies that for $i = 1, 2$,

$$\|\nabla f_i(w_i)\|_2 \geq \mu\|w_i - w_i^*\|.$$

Since $\|\nabla f_i(w_i)\| \leq O(\epsilon''_{\mathcal{A}})$, we get the desired result. □

Since strong convexity is a special case of PL, this theorem implies that if we run enough iterations of a convergent algorithm $\mathcal{A}$ on a λ-strongly convex loss function, then we would expect uniform stability on the order of

$$\epsilon_{stab} = O\left(\frac{L^2}{\lambda n}\right).$$

In particular, this theorem recovers the stability estimates for ERMs and SGD applied to strongly convex functions proved in [12] and [43], respectively.

In order to make this result more generally applicable, we would like to extend the theorem to a larger class of functions than just globally PL functions. If we assume boundedness of the loss function, then we can derive a similar result for globally QG functions. This leads us to the following theorem:

Theorem 8.18. *Assume that for all S, f_S satisfies the QG condition with parameter μ, moreover let Assumption 8.15 hold. Suppose that for all z and $w \in \mathcal{X}$, $\ell(w; z) \leq c$. Then $\mathcal{A}$ is uniformly stable with parameter ϵ_{stab} satisfying:*

Case 1: *If for all S, $\|w_S - w_S^*\| \leq O(\epsilon_{\mathcal{A}})$ then for all z we have:*

$$\epsilon_{stab} \leq O(L\epsilon_{\mathcal{A}}) + 2L\sqrt{\frac{c}{\mu n}}.$$

Case 2: *If for all S, $|f_S(w_S) - f_S^*| \leq O(\epsilon'_{\mathcal{A}})$ then for all z we have:*

$$\epsilon_{stab} \leq O\left(L\sqrt{\frac{\epsilon'_{\mathcal{A}}}{\mu}}\right) + 2L\sqrt{\frac{c}{\mu n}}.$$

To prove this, we will use the following lemma.

Lemma 8.19. *Let f_S be QG and assume that $\ell(w; z) \leq c$ for all z and $w \in \mathcal{X}$. We assume* **A1** *as above. Then for S_1, S_2 differing in at most one place,*

$$\|w_1^* - w_2^*\|_2 \leq 2\sqrt{\frac{c}{\mu n}}.$$

Proof. By QG:

$$\begin{aligned}\frac{\mu}{2}\|w_1^* - w_2^*\|_2^2 &\leq |f_1(w_2^*) - f_1(w_1^*)| \\ &\leq |f_1(w_2^*) - f_2(w_2^*)| + |f_2(w_2^*) - f_1(w_1^*)|\end{aligned}$$

Note that for all w, $|f_1(w) - f_2(w)| = \frac{1}{n}|\ell(w; z_n) - \ell(w; z'_n)|$, so this is bounded by $\frac{c}{n}$. By this same reasoning we get:

$$f_2(w_2^*) \leq f_2(w_1^*) \leq f_1(w_1^*) + \frac{c}{n}$$

The desired result follows. □

Proof of Theorem 8.18. Using the fact that $\ell(\cdot; z)$ is L-Lipschitz we get

$$\begin{aligned}|\ell(w_1; z) - \ell(w_2; z)| &\leq L\|w_1 - w_2\|_2 \\ &\leq L\|w_1 - w_1^*\|_2 + L\|w_1^* - w_2^*\|_2 + L\|w_2^* - w_2\|_2 \qquad (8.12)\end{aligned}$$

Note that by **A1**, we know that w_1^* is the closest optimal point of f_1 to w_2^*. By Lemma 8.19,

$$\|w_1^* - w_2^*\|_2 \leq 2\sqrt{\frac{c}{\mu n}}.$$

This bounds the second term of 8.11. The first and third terms must be bounded differently depending on the case.

Case 1: By assumption, for $i = 1, 2$, $\|w_i - w_i^*\|_2 = O(\epsilon_{\mathcal{A}})$, proving the result.

Case 2: By the QG property, we get that for $i = 1, 2$,

$$\begin{aligned}\|w_i - w_i^*\|_2 &\leq \frac{\sqrt{2}}{\sqrt{\mu}}\sqrt{f_i(w_i) - f_i^*} \\ &= O\left(\sqrt{\frac{\epsilon'_{\mathcal{A}}}{\mu}}\right)\end{aligned}$$

This proves the desired bound. □

Remark 8.20. *By analogous reasoning to that in Remark 8.11, both Theorem 8.17 and Theorem 8.18 hold if you only have information about the output of $\mathcal{A}$ in expectation.*

8.3 Examples of PL Loss Functions

8.3.1 Compositions of Strongly Convex and Piecewise-Linear Functions

As the bounds above show, the PL and QG conditions are sufficient for algorithmic stability and therefore imply good generalization. In this section, we show that the PL condition actually arises in some interesting machine learning setups, including least

squares minimization, strongly convex functions composed with piecewise linear functions, and neural networks with linear activation functions. A first step towards a characterization of PL loss functions was proved by Karimi et al. [48], which established that the composition of a strongly-convex function and a linear function results in a loss that satisfies the PL condition.

We wish to generalize this result to piecewise linear activation functions. Suppose that $\sigma : \mathbb{R} \to \mathbb{R}$ is defined by $\sigma(z) = c_1 z$, for $z > 0$ and $\sigma(z) = c_2 z, for z \leq 0$. Here $c_i > 0$. For a vector $z \in \mathbb{R}^n$, we denote by $\sigma(z) \in \mathbb{R}^n$ the vector whose ith component is $\sigma(z_i)$. Note that this encompasses leaky-ReLU functions. Following similar techniques to those in [48], we get the following result showing that the composition of strongly convex functions with piecewise-linear functions are PL.

Theorem 8.21. *Let g be strongly-convex with parameter λ, σ a leaky ReLU activation function with slopes $c1$ and c_2, and X a matrix with minimum singular value $\sigma_{\min}(X)$. Let $c = \min\{|c_1|, |c_2|\}$. Then $f(w) = g(\sigma(Xw))$ is PL almost everywhere with parameter $\mu = \lambda \sigma_{\min}(X)^2 c^2$.*

Proof. For almost all w, we can write $\sigma(Xw)$ as $\text{diag}(b)Xw$ for a vector b where $b_i(Xw)_i = \sigma((Xw)_i)$ (this only exludes points w such that $(Xw)_i$ is on a cusp of the piecewise-linear function). Then in an open neighborhood of such an w, we find:

$$f(w) = g(\sigma(Xw)) = g(\text{diag}(b)Xw)$$

For a given w, let w_p be the closest global minima of f (*i.e.,* the closest point such

that $f^* = f(w_p)$). By strong convexity of g, we find:

$$\begin{aligned} g(\text{diag}(b)Xw_p) &\geq g(\text{diag}(b)Xw) + \langle \nabla g(\text{diag}(b)Xw), \text{diag}(b)X(w_p - w)\rangle \\ &\quad + \frac{\lambda}{2}\|\text{diag}(b)X(w_p - w)\|_2^2 \\ \implies g(\text{diag}(b)Xw_p) &\geq g(\text{diag}(b)Xw) + \langle X^T\text{diag}(b)^T\nabla g(\text{diag}(b)Xw), w_p - w\rangle \\ &\quad + \frac{\lambda}{2}\|\text{diag}(b)X(w_p - w)\|_2^2 \\ \implies f(w_p) &\geq f(w) + \langle \nabla f(w), w_p - w\rangle + \frac{\lambda\sigma_{\min}(X)^2\sigma_{\min}(\text{diag}(b))^2}{2}\|w_p - w\|_2^2. \end{aligned}$$

Note that the minimum singular value of $\text{diag}(b)$ is the square root of the minimum eigenvalue of $\text{diag}(b)^2$. Since $\text{diag}(b)^2$ has entries c_i^2 on the diagonal, we know that the minimum singular value is at least $c = \min_i\{|c_i|\}$. Therefore we get:

$$\begin{aligned} f(w_p) &\geq f(w) + \langle f(w), w_p - w\rangle + \frac{\lambda\sigma_{\min}(X)^2c^2}{2}\|w_p - w\|_2^2 \\ &\geq f(w) + \min_y\left[\langle \nabla f(w), y - w\rangle + \frac{\lambda\sigma_{\min}(X)^2c^2}{2}\|y - w\|_2^2\right] \\ &= f(w) - \frac{1}{2\lambda\sigma_{\min}(X)^2c^2}\|\nabla f(w)\|_2^2. \end{aligned}$$

□

In particular, 1-layer neural networks with a squared error loss and leaky ReLU activations satisfy the PL condition. More generally, this holds for any piecewise-linear activation function with slopes $\{c_i\}_{i=1}^k$. As long as each slope is non-zero and X is full rank, the result above shows that the PL condition is satisfied.

8.3.2 Linear Neural Networks

The results above only concern one layer neural networks. Given the prevalence of deep networks, we would like to say something about the associated loss function. As it turns

out, we can prove that a PL inequality holds in large regions of the parameter space for deep linear networks. Note that for matrix-valued functions, the correct analog of the PL condition involves a bound on the Frobenius norm of the gradient with respect to the matrix. This allows us to keep the computations in matrix form, which will make the proofs more manageable.

Say we are given a training set $S = \{z_1, \dots, z_n\}$ where $z_i = (x_i, y_i)$ for $x_i, y_i \in \mathbb{R}^d$. Our neural network will have ℓ fully-connected non-input layers, each with d neurons and linear activation functions. We will parametrize the neural network model via $W_1, \dots, W_\ell$, where each $W_i \in \mathbb{R}^{d \times d}$. That is, the output at the first non-input layer is $u_1 = W_1 x$ and the output at layer $k \geq 2$ is $A_k u_{k-1}$. Letting $X, Y \in \mathbb{R}^{d \times N}$ be the matrices with x_i, y_i as their columns (respectively), we can then write our loss function as

$$f(W) = \frac{1}{2}\|W_\ell W_{\ell-1} \dots W_1 X - Y\|_F^2.$$

Let $W = W_\ell W_{\ell-1} \dots W_1$. The optimal value of W is $W^* = YX^+$. Here, $X^+ = X^T(XX^T)^{-1}$ is the pseudoinverse of X. We assume that $X \in \mathbb{R}^{d \times N}$ has rank d so that XX^T is invertible. We will also make use of the following lemma.

Lemma 8.22. *Let $W \in \mathbb{R}^{d \times d}$ be some weight matrix. Then for $C = \|(XX^T)^{-1}X\|_F^2$, we have*

$$C\|(WX - Y)X^T\|_F^2 \geq \|WX - Y\|_F^2 - \|YX^+X - Y\|_F^2.$$

Proof. Using basic properties of the Frobenius norm and the definition of the pseudo-inverse, we have

$$\begin{aligned}\|(WX-Y)X^T\|_F^2\|(XX^T)^{-1}X\|_F^2 &\geq \|(WX-Y)X^T(XX^T)^{-1}X\|_F^2 \\ &= \|(WX-Y)X^+X\|_F^2 \\ &= \|WXX^+X - YX^+X\|_F^2 \\ &= \|WX - YX^+X\|_F^2.\end{aligned}$$

This last step follows by basic properties of the pseudo-inverse. By the triangle inequality,

$$\|WX-Y\|_F^2 = \|WX-YX^+X+YX^+X-Y\|_F^2 \leq \|YX^+X-Y\|_F^2 + \|WX-YX^+X\|_F^2.$$

Note that $YX^+X - Y$ is the component of Y that is orthogonal to the row-space of X, while YX^+X is the projection of Y on to this row space. Therefore, $YX^+X - Y$ is orthogonal to $WX - YX^+X$ with respect to the trace inner product. Therefore, the inequality above is actually an equality, that is

$$\|WX-Y\|_F^2 = \|YX^+X-Y\|_F^2 + \|WX-YX^+X\|_F^2.$$

Putting this all together, we find

$$\|(XX^T)^{-1}X\|_F^2\|(WX-Y)X^T\|_F^2 \geq \|WX-Y\|_F^2 - \|YX^+X-Y\|_F^2.$$

□

For a matrix A, let $\sigma_{\min}(A)$ denote the smallest singular value of A. For a given $W_1, \ldots, W_\ell$, let $W = W_\ell W_{\ell-1} \ldots W_1$. We can then prove the following lemma.

Lemma 8.23. *Suppose that the W_i satisfy $\sigma_{\min}(W_i) \geq \tau > 0$ for all i. Then,*

$$\|\nabla f(W_1, \ldots, W_\ell)\|_F^2 \geq \ell \tau^{2\ell-2} \|(WX - Y)X^T\|_F^2.$$

Proof. Our proof uses similar techniques to that of Hardt and Ma [42]. We wish to compute the gradient of f with respect to a matrix W_j. One can show the following:

$$\frac{\partial f}{\partial W_j} = W_{j+1}^T \ldots W_\ell^T (WX - Y) X^T W_1^T \ldots W_{j-1}^T.$$

Using the fact that for a matrix $A \in \mathbb{R}^{d \times d}$ and another matrix $B \in \mathbb{R}^{d \times k}$, we have $\|AB\|_F \geq \sigma_{\min}(A)\|B\|_F$, we find:

$$\left|\left| \frac{\partial f}{\partial W_j} \right|\right|_F \geq \prod_{i \neq j} \sigma_{\min}(W_i) \|(WX - Y)X^T\|_F.$$

By assumption, $\sigma_{\min}(W_j) \geq \tau$. Therefore:

$$\left|\left| \frac{\partial f}{\partial W_j} \right|\right|_F^2 \geq \tau^{2\ell-2} \|(WX - Y)X^T\|_F^2.$$

Taking the gradient with respect to all W_i we get:

$$\begin{aligned} \left|\left| \frac{\partial f}{\partial (W_1, \ldots, W_\ell)} \right|\right|_F^2 &= \sum_{j=1}^{\ell} \left|\left| \frac{\partial f}{\partial W_j} \right|\right|_F^2 \\ &\geq \ell \tau^{2\ell-2} \|(WX - Y)X^T\|_F^2. \end{aligned}$$

□

Combining Lemmas 8.22 and 8.23, we derive the following interesting corollary about when critical points are global minimizers. This result is not directly related to the work above, but gives an easy way to understand the landscape of critical points of deep linear networks.

Theorem 8.24. *Let $(W_1, \ldots, W_\ell)$ be a critical point such that each W_i has full rank. Then $(W_1, \ldots, W_\ell)$ is a global minimizer of f.*

Proof. Since each W_i has full rank, we know that $\tau = \min_i \sigma_{\min}(W_i) > 0$. Using Lemma 8.23 and the fact that $\nabla f(W_1, \ldots, W_\ell) = 0$, we get $\|(WX - Y)X^T\|_F^2 = 0$. Therefore, $WXX^T = YX^T$. Assuming that (XX^T) is invertible, we find that $W = YX^+$, which equals W^*. □

Thematically similar results have been derived previously for 1 layer networks in [99] and for deep neural networks in [106]. In [49], Kawaguchi derives a similar result to ours for deep linear neural networks. Kawaguchi shows that every critical point is either a global minima or a saddle point. Our result, by contrast, implies that all full-rank critical points are global minima.

Lemmas 8.22 and 8.23 can also be combined to show that linear networks satisfy the PL condition in large regions of parameter space, as the following theorem says.

Theorem 8.25. *Suppose our weight matrices $(W_1, \ldots, W_\ell)$ satisfy $\sigma_{\min}(W_i) \geq \tau$ for $\tau > 0$. Then $f(W_1, \ldots, W_\ell)$ satisfies the following PL inequality:*

$$\frac{1}{2}\left|\left|\frac{\partial f}{\partial (W_1, \ldots, W_\ell)}\right|\right|_F^2 \geq \frac{\ell \tau^{2\ell-2}}{\|(XX^T)^{-1}X\|_F^2}(f(W_1, \ldots, W_\ell) - f^*).$$

Proof. By Lemma 8.23 and Lemma 8.22 we find:

$$\begin{aligned}
\frac{1}{2}\left|\left|\frac{\partial f}{\partial (W_1, \ldots, W_\ell)}\right|\right|_F^2 &\geq \ell \tau^{2\ell-2}\frac{1}{2}\|(WX - Y)X^T\|_F^2 \\
&\geq \ell \tau^{2\ell-2}\frac{1}{\|(XX^T)^{-1}X\|_F^2}\frac{1}{2}(\|WX - Y\|_F^2 - \|YX^+X - Y\|_F^2) \\
&= \frac{\ell \tau^{2\ell-2}}{\|(XX^T)^{-1}X\|_F^2}(f(W_1, \ldots, W_\ell) - f^*).
\end{aligned}$$

□

In other words, taking

$$\mu = \frac{{\ell_T}^{2\ell-2}}{\|(XX^T)^{-1}X\|_F^2},$$

then at every point $(W_1, \ldots, W_\ell)$ satisfying $\sigma_{\min}(W_i) \geq \tau$, the loss function of our linear network satisfies the PL condition with parameter μ.

Chapter 9

Stability of Some First-order Methods

Recipe 12: Ginger Peach Pie

Ingredients

- Pie dough, enough for 2 crusts (see Recipe 1)
- 3 $\frac{1}{2}$ pounds peaches
- 1 lemon
- $\frac{1}{4}$ cup sugar
- $\frac{1}{4}$ cup brown sugar
- 3 tablespoon cornstarch
- $\frac{1}{4}$ teaspoon cinnamon
- Nutmeg, to taste
- $\frac{1}{4}$ teaspoon salt
- 1 heaping teaspoon ground ginger
- 1 egg yolk

Preparation

1. Preheat oven to 425° F. Roll one dough out and transfer to a 9-inch pie pan. Trim overhang and return to fridge.
2. Halve and pit the peaches, and then cut into $\frac{1}{3}$-inch slices. Add to a large bowl with juice of $\frac{1}{2}$ of the lemon and a $\frac{1}{2}$ teaspoon of lemon zest.
3. In a small dish, stir together remaining dry ingredients. Add to bowl of

peaches and toss to coat evenly.

4. Scoop filling into the bottom pie dough including juices. Roll out top dough and add decorative vents (or make a lattice crust). Mix egg yolk with a teaspoon of water and brush crust. Sprinkle with sugar.

5. Bake for 20 minutes until the crust begins to brown. Reduce temperature to 375° F and bake for another 30-40 minutes, until filling is bubbling. Loosely cover with foil if the crust is ever browning too quickly.

9.1 Stability for Strongly Convex and PL Loss Functions

We wish to apply our bounds from the previous section to popular convergent gradient-based methods. We consider SGD, GD, RCD, and SVRG. When we have L-Lipschitz, μ-PL loss functions f_S and n training examples, Theorem 8.17 states that any learning algorithm $\mathcal{A}$ has uniform stability ϵ_{stab} satisfying

$$\epsilon_{stab} \leq O\left(L\sqrt{\frac{\epsilon_{\mathcal{A}}}{\mu}}\right) + O\left(\frac{L^2}{\mu n}\right).$$

Here, $\epsilon_{\mathcal{A}}$ refers to how quickly $\mathcal{A}$ converges to the optimal value of the loss function. Specifically, this holds if the algorithm produces a model w_S satisfying $|f_S(w_s) - f_S^*| \leq O(\epsilon_{\mathcal{A}})$. For example, if we want to guarantee that our algorithm has the same stability as SGD in the strongly convex case, then we need to determine how many iterations T

we need to perform such that

$$\epsilon_{\mathcal{A}} = O\left(\frac{L^2}{\mu n}\right). \tag{9.1}$$

The convergence rates of SGD, GD, RCD, and SVRG have been studied extensively in the literature [16, 46, 48, 68, 69]. The results are given below. We assume that the loss function is β-smooth. When necessary to state the result, we assume a constant step-size of γ. Figure 12 below summarizes the values of $\epsilon_{\mathcal{A}}$ for T iterations of SGD, GD, RCD, and SVRG applied to λ-strongly convex loss functions and μ-PL loss functions. For RCD, we assume that the gradient is coordinate-wise β-Lipschitz and that the feature vectors x are d-dimensional. For SVRG, we assume that we compute a full gradient after every m iterations of stochastic gradient updates.

Note that if Eq. (9.1) holds, then Corollary 8.14 implies that our algorithm is uniformly stable with parameter $\epsilon_{stab} = O(L^2/\mu n)$. Moreover, this is the same stability as that of SGD for strongly-convex functions [43], and saddle point avoiding algorithms on strict-saddle loss functions [39].

We use the above convergence rates of these algorithms in the λ-strongly convex and μ-PL settings to determine how many iterations are required such that we get stability that is $O(L^2/\mu n)$. The results are summarized in Figure 13 below.

Note that in the μ-PL case, although it is a non-convex setup the above algorithms all exhibit the same stability for these values of T. This is not the case in general. Several studies have observed that small-batch SGD offers superior generalization performance compared to large-batch SGD and full-batch GD when training deep neural networks [50].

	λ-SC	μ-PL
SGD	$O\left((1-2\gamma\lambda)^T + \frac{\gamma\beta^2}{2\lambda}\right)$ [16]	$O\left((1-2\gamma\mu)^T + \frac{\gamma\beta^2}{2\mu}\right)$ [48]
GD	$O\left(\left(1-\frac{\lambda}{\beta}\right)^T\right)$ [69]	$O\left(\left(1-\frac{\mu}{\beta}\right)^T\right)$ [48]
RCD	$O\left(\left(1-\frac{\lambda}{d\beta}\right)^T\right)$ [68]	$O\left(\left(1-\frac{\mu}{d\beta}\right)^T\right)$ [48]
SVRG	$O\left(\left(\frac{1}{\lambda\gamma(1-2\beta\gamma)m} + \frac{2\beta\gamma}{1-2\beta\gamma}\right)^T\right)$ [46]	$O\left(\left(\frac{1}{\mu\gamma(1-2\beta\gamma)m} + \frac{2\beta\gamma}{1-2\beta\gamma}\right)^T\right)$ [48]

Figure 12: Convergence rates for T iterations of various gradient-based algorithms in the λ-SC and μ-PL settings.

9.2 Stability of Gradient Descent for Convex Loss Functions

In [43], Hardt et al. proved bounds on the uniform stability of SGD. They also noted that GD does not appear to be provably as stable for the non-convex case and sketched a situation in which this difference would appear. Due to the similarity of SGD and GD, one may expect similar uniform stability. This holds for convex loss functions, setting as we show below.

In [43], Hardt et al. show the following theorem.

Theorem 9.1 ([43])**.** *Let $\ell(\cdot; z)$ be L-Lipschitz, β-smooth, and convex for all z. Say we perform T iterations of SGD with a constant step-size $\gamma \leq 2/\beta$ to train iterates w_t on S*

	λ-SC	μ-PL
SGD	$O\left(\frac{\beta n}{L^2 \lambda}\right)$	$O\left(\frac{\beta n}{L^2 \mu}\right)$
GD	$O\left(\frac{\log\left(\frac{L}{\lambda n}\right)}{\log\left(1-\frac{\lambda}{\beta}\right)}\right)$	$O\left(\frac{\log\left(\frac{L}{\mu n}\right)}{\log\left(1-\frac{\mu}{\beta}\right)}\right)$
RCD	$O\left(\frac{\log\left(\frac{L}{\lambda n}\right)}{\log\left(1-\frac{\lambda}{d\beta}\right)}\right)$	$O\left(\frac{\log\left(\frac{L}{\mu n}\right)}{\log\left(1-\frac{\mu}{d\beta}\right)}\right)$
SVRG	$O\left(\frac{\log\left(\frac{L}{\lambda n}\right)}{\log\left(\frac{1}{\lambda\gamma(1-2\beta\gamma)m}+\frac{2\beta\gamma}{1-2\beta\gamma}\right)}\right)$	$O\left(\frac{\log\left(\frac{L}{\mu n}\right)}{\log\left(\frac{1}{\mu\gamma(1-2\beta\gamma)m}+\frac{2\beta\gamma}{1-2\beta\gamma}\right)}\right)$

Figure 13: The number of iterations T that achieves stability $\epsilon_{stab} = O(L^2/\mu n)$ for various gradient-based algorithms with step-size γ in the λ-SC and μ-PL settings.

and $\hat{w}_t$ on S'. Then for all such S, S' with $|S| = |S'| = n$ such that S, S' differ in at most one example,

$$\mathbb{E}_{\mathcal{A}}[\|w_T - \hat{w}_T\|_2] \leq \frac{2\gamma L T}{n}$$

If $\ell(\cdot; z)$ is λ-strongly convex for all z, then

$$\mathbb{E}_{\mathcal{A}}[\|w_T - \hat{w}_T\|_2] \leq \frac{2L}{\lambda n}$$

Performing similar analysis for gradient descent, we obtain the following theorem.

Theorem 9.2. *Assume that for all z, $\ell(\cdot; z)$ is convex, β-smooth, and L-Lipschitz. Say we run GD for T iterations with step-sizes γ_t such that $\gamma_t \leq \frac{2}{\beta}$. Then GD is uniformly*

stable with

$$\epsilon_{stab} \leq \frac{2L^2}{n} \sum_{t=0}^{T} \gamma_t.$$

If $\ell(\cdot; z)$ is λ-strongly convex for all z, then GD is uniformly stable with

$$\epsilon_{stab} \leq \frac{2L}{\lambda n}.$$

To prove this theorem, we use similar techniques to those in [43]. We first consider the convex case.

Proof. By direct computation, we have:

$$\begin{aligned}
\|w_T - \hat{w}_T\|_2 = \|w_{T-1} - \hat{w}_{T-1} - \frac{\gamma_T}{n} \sum_{i=1}^{n-1} \Big(\nabla \ell(w_{T-1}; z_i) - \nabla \ell(\hat{w}_{T-1}; z_i) \Big) \\
- \frac{\gamma_T}{n} \nabla \ell(w_{T-1}; z_n) + \frac{\gamma_T}{n} \nabla \ell(\hat{w}_{T-1}; z_n')\|_2 \\
\leq \|w_{T-1} - \hat{w}_{T-1} - \frac{\gamma_T}{n} \sum_{i=1}^{n-1} \Big(\nabla \ell(w_{T-1}; z_i) - \nabla \ell(\hat{w}_{T-1}; z_i) \Big) \|_2 \\
+ \|\frac{\gamma_T}{n} \nabla \ell(w_{T-1}; z_n) - \frac{\gamma_T}{n} \nabla \ell(\hat{w}_{T-1}; z_n')\|_2.
\end{aligned}$$

Note that since $\ell(\cdot; z)$ is L-Lipschitz, the second part of this summand is bounded by $\frac{2\gamma_T L}{n}$. We now wish to bound the first part. We get:

$$
\begin{aligned}
&\|w_{T-1} - \hat{w}_{T-1} - \frac{\gamma_T}{n}\sum_{i=1}^{n-1}\Big(\nabla\ell(w_{T-1};z_i) - \nabla\ell(\hat{w}_{T-1};z_i)\Big)\|_2^2 \\
&= \|w_{T-1} - \hat{w}_{T-1}\|_2^2 - \frac{2\gamma_T}{n}\langle w_{T-1} - \hat{w}_{T-1}, \sum_{i=1}^{n-1}\Big(\nabla\ell(w_{T-1};z_i) - \nabla\ell(\hat{w}_{T-1};z_i)\Big)\rangle \\
&\qquad + \frac{\gamma_T^2}{n^2}\|\sum_{i=1}^{n-1}\Big(\nabla\ell(w_{T-1};z_i) - \nabla\ell(\hat{w}_{T-1};z_i)\Big)\|_2^2 \\
&\le \|w_{T-1} - \hat{w}_{T-1}\|_2^2 - \sum_{i=1}^{n-1}\left(\frac{2\gamma_T}{n^2}\langle w_{T-1} - \hat{w}_{T-1}, \nabla\ell(w_{T-1};z_i) - \nabla\ell(\hat{w}_{T-1};z_i)\rangle\right) \\
&\qquad + \sum_{i=1}^{n-1}\left(\frac{\gamma_T^2}{n^2}\|\nabla\ell(w_{T-1};z_i) - \nabla\ell(\hat{w}_{T-1};z_i)\|_2^2\right) \\
&\le \|w_{T-1} - \hat{w}_{T-1}\|_2^2 + \sum_{i=1}^{n-1}(\frac{\gamma_T^2}{n^2} - \frac{2\gamma_T}{n^2\beta})\|\nabla\ell(w_{T-1};z_i) - \nabla\ell(\hat{w}_{T-1};z_i)\|_2^2.
\end{aligned}
$$

Note that this last step holds by co-coercivity of the gradient of $\ell(\cdot, z_i)$. In particular, if $\gamma_T \le \frac{2}{\beta}$, each of the $n-1$ summands on the right will be nonpositive. Therefore for such γ_T, we get:

$$
\|w_{T-1} - \hat{w}_{T-1} - \frac{\gamma_T}{n}\sum_{i=1}^{n-1}\Big(\nabla\ell(w_{T-1};z_i) - \nabla\ell(\hat{w}_{T-1};z_i)\Big)\|_2 \le \|w_{T-1} - \hat{w}_{T-1}\|_2.
$$

So, if $\frac{\gamma_t}{n} \le \frac{2}{\beta}$ for all t, we get:

$$
\begin{aligned}
\|w_T - \hat{w}_T\|_2 &\le \|w_{T-1} - \hat{w}_{T-1}\|_2 + \frac{2\gamma_T L}{n} \\
&\le \|w_{T-2} - \hat{w}_{T-2}\|_2 + \frac{2L}{n}(\gamma_{T-1} + \gamma_T) \\
&\vdots \\
&\le \|w_0 - \hat{w}_0\|_2 + \frac{2L}{n}\sum_{t=1}^{T}\gamma_t \\
&= \frac{2L}{n}\sum_{t=1}^{T}\gamma_t.
\end{aligned}
$$

This last step follows from the fact that $w_0 = \hat{w}_0$ if we initialize at the same point. Using the fact that $f(w)$ is L-Lipschitz (since ℓ is), we get:

$$|f(w_T; z) - f(\hat{w}_T; z)| \leq \frac{2L^2}{n} \sum_{t=0}^{n} \gamma_t.$$

□

We now move to the λ-strongly convex case. For simplicity of analysis, we assume that we use a constant step-size γ such that $\gamma \leq 1/\beta$.

Proof. The proof remains the same, except when using co-coercivity. Under this assumption, some plug and play in an analogous fashion will show:

$$\begin{aligned}
&\|w_{T-1} - \hat{w}_{T-1} - \frac{\gamma_T}{n} \sum_{i=1}^{n-1} \Big(\nabla \ell(w_{T-1}; z_i) - \nabla \ell(\hat{w}_{T-1}; z_i) \Big) \|_2^2 \\
&\leq \left(1 - \frac{2\gamma\lambda\beta}{\lambda + \beta}\right) \|w_{T-1} - \hat{w}_{T-1}\|_2^2 \\
&\qquad + \sum_{i=1}^{n-1} (\frac{\gamma_T^2}{n^2} - \frac{2\gamma_T}{n\beta}) \|\nabla \ell(w_{T-1}; z_i) - \nabla \ell(\hat{w}_{T-1}; z_i)\|_2^2.
\end{aligned}$$

Note that if $\gamma \leq \frac{1}{\beta}$ then the second term is nonnegative and one can show that this implies:

$$\begin{aligned}
&\|w_{T-1} - \hat{w}_{T-1} - \frac{\gamma_T}{n} \sum_{i=1}^{n-1} \Big(\nabla \ell(w_{T-1}; z_i) - \nabla \ell(\hat{w}_{T-1}; z_i) \Big) \|_2 \\
&\leq \Big(1 - \gamma\lambda\Big) \|w_{T-1} - \hat{w}_{T-1}\|_2.
\end{aligned}$$

Combining, this shows:

$$\begin{aligned}
\|w_T - \hat{w}_{T-1}\|_2 &\leq (1-\gamma\lambda)\|w_{T-1} - \hat{w}_{T-1}\|_2 + \frac{2L\gamma}{n} \\
&\leq (1-\gamma\lambda)^2\|w_{T-2} - \hat{w}_{T-2}\|_2 + \frac{2L\gamma}{n}\Big(1 + (1-\gamma\lambda)\Big) \\
&\quad\vdots \\
&\leq \frac{2L\gamma}{n}\sum_{t=0}^{T}(1-\gamma\lambda)^t \\
&\leq \frac{2L}{\lambda n}.
\end{aligned}$$

This implies uniform stability with parameter $\frac{2L^2}{\lambda n}$. □

9.3 Instability of Gradient Descent for Non-convex Loss Functions

This similarity breaks down in the non-convex setting. Below we construct an explicit example where GD is not uniformly stable, but SGD is. This example formalizes the intuition given in [43].

For $x, w \in \mathbb{R}^m$ and $y \in \mathbb{R}$, we let

$$\ell(w;(x,y)) = (\langle w, x\rangle^2 + \langle w, x\rangle - y)^2.$$

Intuitively, this is a generalized quadratic model where the predicted label $\hat{y}$ for a given x is given by

$$\hat{y} = \langle w, x\rangle^2 + \langle w, x\rangle.$$

The above predictive model can be described by the following 1-layer neural network using a quadratic and a linear activation function, denoted z^2 and z.

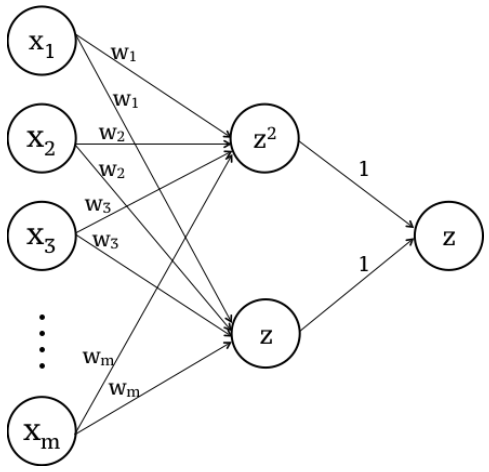


Figure 14: A neural network representing a generalized quadratic model.

Note that for a fixed x, y, $\ell(w;(x,y))$ is non-convex, *i.e.,* it is a quartic polynomial in the weight vector w. We will use this function and construct data sets S, S' that differ in only one entry, for which GD produces significantly different models. We consider this loss function for all $z = (x, y)$ with $\|z\|_2 \leq C$ for C sufficiently large. When $m = 1$, the loss function simplifies to

$$\ell(w;(x,y)) = (w^2x^2 + wx - y)^2.$$

Consider $\ell(w;(x,y))$ at $(-1,1)$ and $(\frac{-1}{2},1)$. Their graphs are as follows. We also graph $g(w)$ which we define as

$$g(w) = \frac{1}{2}\Big(\ell\big(w;(-1,1)\big) + \ell\big(w;(\frac{-1}{2},1)\big)\Big).$$

Note that the last function has two distinct basins of different heights. Taking the gradient, one can show that the right-most function in Figure 15 has zero slope at $\hat{w} \approx 0.598004$. Comparing $\ell(w;(-1,1))$ and $\ell(w;(\frac{-1}{2},1))$, we see that the sign of their slopes agrees on $(-\frac{1}{2},\frac{1}{2})$ and on $(1,\frac{3}{2})$. The slopes are of different sign in the interval $[\frac{1}{2},1]$. We will use this to our advantage in showing that gradient descent is not stable,

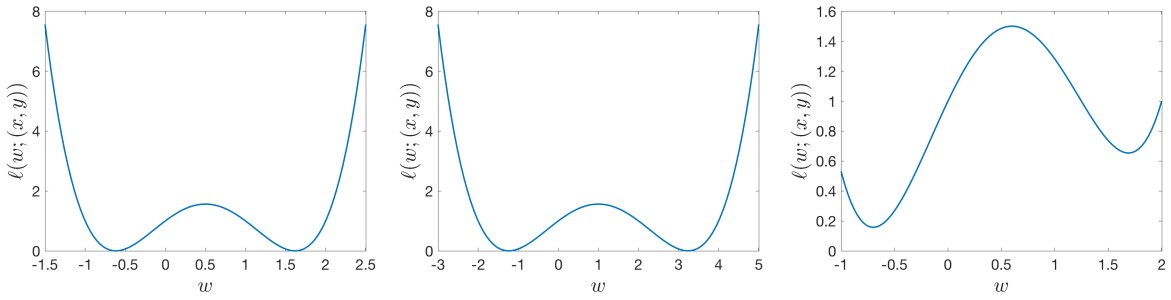


Figure 15: Graphs of the functions $\ell(w;(-1,1))$ (left), $\ell(w;(\frac{-1}{2},1))$ (middle), and $g(w) = \frac{1}{2}[\ell(w;(-1,1)) + \ell(w;(\frac{-1}{2},1))]$ (right).

while that SGD is.

We will construct points (x_1, y_1) and (x_2, y_2) such that $\ell(w;(x_1,y_1))$ and $\ell(w;(x_2,y_2))$ have positive and negative slope at $\hat{w}$. To do so, we will first construct an example with a slope of zero at $\hat{w}$. A straightforward computation shows that $\ell(w;(\frac{-1}{2\hat{w}},0))$ has slope zero at $\hat{w}$. A graph of this loss function is given below. Note that $\hat{w}$ corresponds to the concave-down critical point in between the two global minima.

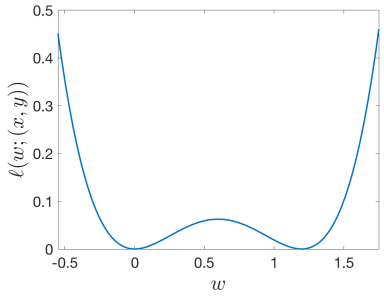


Figure 16: Graph of the function $\ell(w;(-\frac{1}{2\hat{w}}),0)$. By construction, this function has critical points at $w = 0, \hat{w}, 2\hat{w}$.

Define

$$z_{\pm} = \left(\frac{-1}{2(\hat{w} \pm \epsilon)}, 0\right).$$

Straightforward calculations show that $\ell(w;(z_+,0))$ will have positive slope for $w \in (0, \hat{w} + \epsilon)$, while $\ell(w;(z_-,0))$ will have negative slope for $w \in (\hat{w} - \epsilon, 2(\hat{w} - \epsilon))$. In

particular, their slopes have opposite signs in the interval $(\hat{w} - \epsilon, \hat{w} + \epsilon)$. Define

$$S = \{z_1, \ldots, z_{n-1}, z_-\}$$
$$S' = \{z_1, \ldots, z_{n-1}, z_+\},$$

where $z_i = (-1, 1)$ for $1 \leq i \leq \frac{n-1}{2}$, $z_i = (-1/2, 1)$ for $\frac{n-1}{2} < i \leq n-1$. By construction, we have

$$f_S(w) = \frac{n-1}{n} g(w) + \ell\left(w; \left(\frac{-1}{2(\hat{w} + \epsilon)}, 0\right)\right).$$
$$f_{S'}(w) = \frac{n-1}{n} g(w) + \ell\left(w; \left(\frac{-1}{2(\hat{w} - \epsilon)}, 0\right)\right).$$

Then $f_S(w)$, $f_{S'}(w)$ will approximately have the shape of the right-most function in Figure 15 above. However, recall that $\frac{d}{dw} g(w) = 0$ at $w = \hat{w}$. Therefore, there is some δ, with $0 < \delta < \epsilon$, such that for all $w \in (\hat{w} - \delta, \hat{w} + \delta)$,

$$\frac{d}{dw} f_S(w) < 0 < \frac{d}{dw} f_{S'}(w). \tag{9.2}$$

Now say that we initialize gradient descent with some step-size $\gamma > 0$ in the interval $(w - \delta, w + \delta)$. By (9.2), the first step of gradient descent on f_S will produce a step moving to the left, but a step moving to the right for $f_{S'}$. This will hold for all $\gamma > 0$. Moreover, the iterations for f_S will continue to move to towards the left basin, while the iterations for $f_{S'}$ will continue to move to the right basin since $\ell(w; (z_+, 0))$ has positive slope for $w \in (0, \hat{w} + \epsilon)$ while $\ell(w; (z_-, 0))$ has negative slope for $w \in (\hat{w} - \epsilon, 2(\hat{w} - \epsilon)$, and that $g(w)$ has positive slope for $w \in (-\frac{1}{2}, \hat{w})$ and negative slope for $w \in (\hat{w}, \frac{3}{2})$.

After enough iterations of gradient descent on $f_S, f_{S'}$, we will obtain models w_S and $w_{S'}$ that are close to the distinct local minima in the right-most graph in Figure 15. This will hold as long as γ is not extremely large, in which case the steps of gradient

descent could simply jump from one local minima to another. To ensure this does not happen, we restrict to $\gamma \leq 1$.

Let $w_1 < w_2$ denote the two local minima of $g(w)$. For $z^* = (-1/2, 1)$, plugging these values into $\ell(w; z^*)$ shows that $|\ell(w_1; z^*) - \ell(w_2; z^*)| > 1$. Since w_S is close to w_1 and $w_{S'}$ is close to w_2, we get the following theorem.

Theorem 9.3. *For all n, for all step-sizes $0 < \gamma \leq 1$, there is a K such that for all $k \geq K$, there are data sets S, S' of size n differing in one entry and a non-zero measure set of initial starting points such that if we perform k iterations of gradient descent with step-size γ on S and S' to get outputs $\mathcal{A}(S), \mathcal{A}(S')$ then there is a z^* such that*

$$|\ell(\mathcal{A}(S); z^*) - \ell(\mathcal{A}(S'); z^*)| \geq \frac{1}{2}.$$

Theorem 9.3 establishes that there exist simple non-convex settings, for which the uniform stability of gradient descent does not decrease with n. In light of the work in [43], where the authors show that for very conservative step-sizes, SGD is stable on non-convex loss function, we might wonder whether SGD is stable in this setting with moderate step-sizes. We know that gradient descent is not stable, by Theorem 9.3, for $\gamma = 1$. For simplicity of analysis, we focus on the case where $\gamma = 1$.

Suppose we run SGD on the above $f_S, f_{S'}$ with step-size 1 and initialize near $\hat{w}$ (we will be more concrete later about where we initialize). With probability $\frac{n-1}{n}$, the first iteration of SGD will use the same example for both S and S', either $z = (-1, 1)$ or $z = (-1/2, 1)$. Computing derivatives at $\hat{w}$ shows

$$\frac{d}{dw}\ell(w; (-1, 1))\bigg|_{w=\hat{w}} \approx -0.486254.$$

$$\frac{d}{dw}\ell(w; (-\frac{1}{2}, 1))\bigg|_{w=\hat{w}} \approx 0.486254.$$

In both cases, the slope is at least 0.4. Therefore, there is some η such that for all $w \in [\hat{w} - \eta, \hat{w} + \eta]$,

$$\frac{d}{dw}\ell(w; (-1, 1)) < -0.4.$$
$$\frac{d}{dw}\ell(w; (-\frac{1}{2}, 1)) > 0.4.$$

Since we are taking $\gamma = 1$ and $\hat{w} \approx 0.598004$, this implies that with probability $\frac{n-1}{n}$, after one step of SGD we move outside of the interval $(0.5, 1)$. This is important since this is the interval where $\ell(w; (-1, 1))$ and $\ell(w; (-1/2, 1))$ have slopes that point them towards distinct basins. Similarly, outside of this interval we also have that the slopes of $\ell(w; (z_{\pm}, 0))$ point towards the same basin.

Therefore, continuing to run SGD in this setting, even if we now decrease the step-size, will eventually lead us to the same basins of $f_S(w), f_{S'}(w)$. Let $w_S, w_{S'}$ denote the outputs of SGD in this setting after enough steps so that we get convergence to within $\frac{1}{n}$ of a local minima. If our first sample z was $(-1, 1)$, we will end up in the right basin, while if our first sample z was $(-1/2, 1)$, we will end up in the left basin. In particular, for ϵ small, $z_{\pm}$ are close enough that the minima of $f_S, f_{S'}$ are within $\frac{1}{n}$ of each other. Note that the minima w_1, w_2 that $w_S, w_{S'}$ are converging to are different. However, because they are in the same basin we know that for ϵ small, $z_{\pm}$ are close enough that $\|w_S - w_{S'}\|_2 \leq O(\frac{1}{n})$. Therefore, we have

$$\|w_S - w_{S'}\|_2 \leq O\left(\frac{1}{n}\right).$$

Note that in our proof of the instability of gradient descent, we only needed to look at z satisfying $\|z\|_2 \leq 2$ to see the instability. However, for SGD if we restrict to $\|z\|_2 \leq 2$, then by compactness we know that $\ell(w; z)$ will be Lipschitz. Therefore, with probability

$\frac{n-1}{n}$,

$$\|\ell(w_S; z) - \ell(w_{S'}; z)\|_2 \leq L\|w_S - w_{S'}\|_2 \leq O\left(\frac{1}{n}\right).$$

With probability $\frac{1}{n}$, SGD first sees the example on which S, S' differ. In this case, $w_S, w_{S'}$ may end up in different basins of the right-most graph in Figure 15. Restricting to $\|z\|_2 \leq 2$, by compactness we have $|\ell(w_S; z) - \ell(w_{S'}; z)| \leq C$ for some constant C. Therefore, $|\ell(w_S; z) - \ell(w_{S'}; z)| = O(1/n)$ with probability $1 - 1/n$ and $|\ell(w_S; z) - \ell(w_{S'}; z)| = O(1)$ with probability $1/n$. This implies the following theorem.

Theorem 9.4. *Suppose that we initialize SGD in $[\hat{w}-\eta, \hat{w}+\eta]$ with a step-size of $\gamma = 1$. Let $\mathcal{A}(S), \mathcal{A}(S')$ denote the output of SGD after k iteration for sufficiently large k. For $\|z\|_2 \leq 2$,*

$$\mathbb{E}_{\mathcal{A}}\big[\ell(\mathcal{A}(S); z) - \ell(\mathcal{A}(S'); z)\big] \leq O\left(\frac{1}{n}\right).$$

This is in stark contrast to gradient descent, which is unstable in this setting. While work in [105] suggests that this stability of SGD even in non-convex settings is a more general phenomenon, proving that this holds remains an open problem.

Part IV

Distributed Machine Learning and Gradient Coding

Chapter 10

Gradient Coding

Recipe 13: Hearty Wheat Bread

Ingredients

- 1 $\frac{1}{4}$ lukewarm water
- 2 teaspoons active dry yeast
- 1 cup whole milk
- $\frac{1}{4}$ cup honey
- 2 tablespoons neutral oil
- 2 $\frac{3}{4}$ cups all-purpose flour
- 2 $\frac{3}{4}$ cups wheat flour
- 1 tablespoon salt

Preparation

1. Pour water in to a bowl and sprinkle yeast on top. Mix in milk, honey, and oil.
2. Add two cups all-purpose flour and salt, stirring to combine. Add the remaining all-purpose and wheat flours. Stir to form a shaggy dough. Let stand for 20 minutes.
3. Knead dough until it forms a ball without sagging, adding flour if sticky. Clean out bowl and coat with oil. Place dough in bowl and cover with plastic wrap. Let dough rise for 1 $\frac{1}{2}$ hours.
4. Divide dough in two and shape each half into a loose ball. Let rest for 10 minutes.
5. Grease two 8×4 loaf pans. Shape

dough balls into loaves and place in pans. Let rise for 30 minutes.

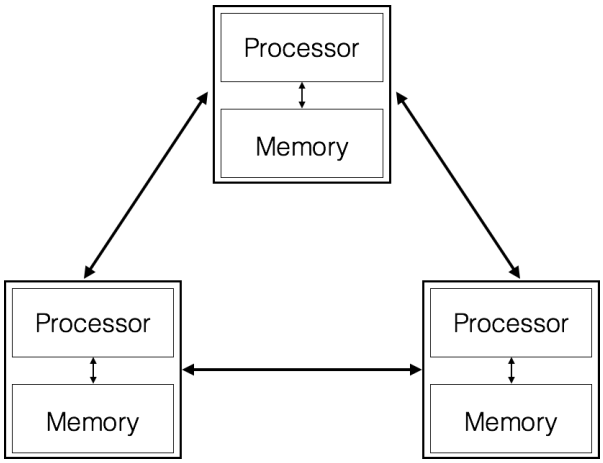
6. Heat oven to 425° F. Slash tops of risen loaves with a serrated knife and place in oven. Turn down heat to 375° F and bake for 30 minutes or until the tops are a dark golden-brown color.

10.1 Background

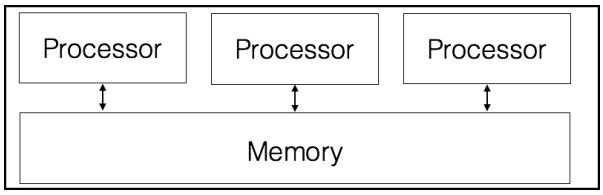
10.1.1 Distributed Computation and the Straggler Effect

Over the last decade, more and more computation is being shifted towards *distributed systems*. Such systems contain multiple computers that communicate with one another and run local computations in order to complete some task. We will refer to such computers as *compute nodes*. We generally represent these systems as a *graph*, where we have vertices connected via edges. The compute nodes form the vertices, and the edges represent the other nodes with which a given node can communicate. Additionally, there are *parallel systems*, in which we have multiple computers, but they have a shared memory. By contrast, in distributed systems each compute node only has its own local memory. If we want to share information between nodes, then we have to send messages between the computers. Figures 17a and 17b below illustrate the difference between these setups.

In this chapter, we will focus on distributed computation. In particular, we will consider a *master-worker* setup. In such setups, there is a single *master node* that assigns tasks to each of the compute nodes. The compute nodes each complete their



(a) A distributed system with three compute nodes.



(b) A parallel system with three compute nodes.

tasks locally, and send them back to the master node.

For example, suppose that we have a large collection of votes from a recent election. We could get one person to look through the votes and count how many went to each candidate. This would be unbearably slow. Instead, we have a coordinator who assigns some number of votes to each person working for them. These workers then add up how many votes went to each candidate. The coordinator can then amalgamate the results and determine who won the election.

This setup is particularly important due to its relative simplicity and its appearance in practical distributed systems. Many modern distributed systems such as ApacheSpark [103] and MapReduce [24] can often be considered as a master-worker setup. A pictorial

representation of a master-worker setup is given in Figure 18.

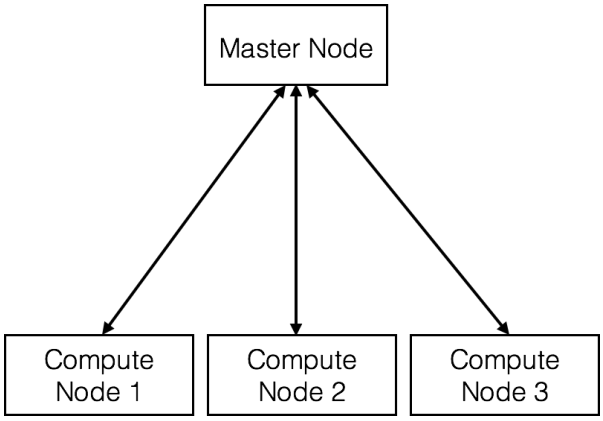


Figure 18: A master-worker with three compute nodes.

In practice, the advent of big data means that we often have access to enormous amounts of data when trying to solve a given problem. As a result, distributed systems have become the de facto choice for scaling out computations to these massive data sets. There are large amounts of theoretical work that quantify how much faster a distributed system is than a single computer. Of course, this only says how much faster the distributed system is theoretically. In practice, distributed systems generally compute tasks faster than single computers, but not as fast as existing theory dictates they should [23, 76]. This commonly observed behavior is referred to as the *speedup saturation phenomenon*. Speedup saturation has many causes. These include the time it takes compute nodes to communicate, shared computing resources between nodes, maintenance activities on the network, and physical issues such as hardware failure and limits on power among the entires system [23, 100]. All these factors often cause some of the compute nodes to take significantly more time to finish their computations than expected. This leads to the presence of *straggler nodes*. These are compute nodes that are substantially slower than the average in the system. We would like to design

distributed algorithms in a way that we do not have to wait for straggler nodes to finish their computations.

10.1.2 Distributed Machine Learning

Due to the large amounts of data used in modern machine learning frameworks, there has been enormous interest in finding ways to perform machine learning in a distributed manner. This can be done in a *synchronous* or an *asynchronous manner*. In the former, we wait for all of the compute nodes to finish and then assign them the next round of tasks. In the asynchronous version, as soon as a compute node finishes, we assign it a new task, whether or not the rest of the nodes are done.

To train machine learning algorithms, we typically use so-called *gradient methods*. For the purposes of this work, there are two salient points about these algorithms. First, they all involve the computation of *gradients*, which measure the accuracy of a machine learning model. By updating the model according to the gradients, we can improve its accuracy. Second, in most applications we have to compute one gradient for each data point. If we are training a computer to do medical diagnosis on a population of patients, we would need to compute one gradient per patient to update our model. In practice, we may have to update the model thousands or millions of times, so we would like to reduce the amount of time it takes to compute these gradients.

Suppose that we have a master-worker system and would like to use a gradient method to train a machine learning model. If we have k training examples, then each step of our gradient method will require k gradient computations. If we have n compute nodes, then we can task each compute node with $\frac{k}{n}$ gradient computations. The compute

nodes then sends their computed gradients to the master, which updates the machine learning model accordingly. The algorithm we just described is one step of *synchronous gradient descent*, where the synchronous part refers to the fact that we wait until all nodes are done computing before we update the model. This kind of idea is relevant to many popular gradient methods, such as mini-batch stochastic gradient descent.

The runtime of synchronous gradient descent is dictated by the runtime of the slowest compute node. If one of the compute nodes in our system takes an hour to compute its gradients, while the rest only take a minute, we still need to wait for the entire hour. In the language of the preceding section, stragglers can greatly impact the runtime.

For example, suppose we have a master-worker system with 4 compute nodes and 4 gradients to compute. As a first step, we could assign 1 gradient to each compute node. We would then have to wait for all 4 nodes to finish to find all the gradients. A pictorial representation of this system is given in Figure 19 below.

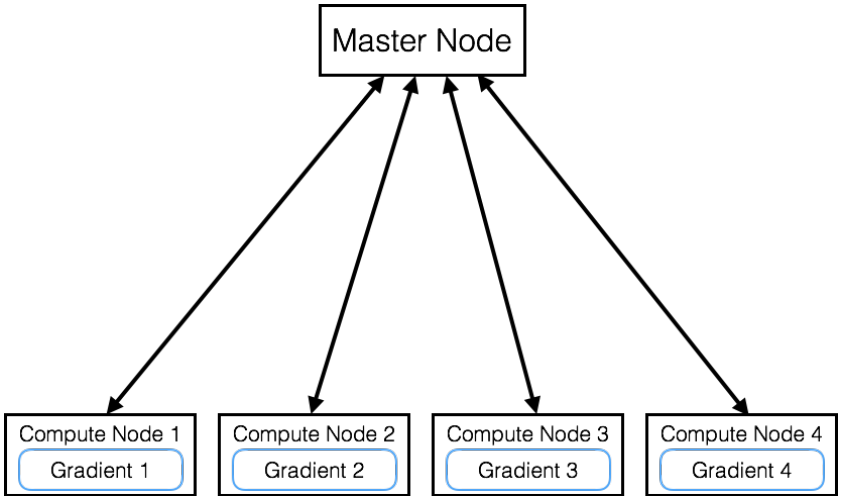


Figure 19: A master-worker system with 4 workers and 1 gradient per worker.

We could instead assign gradients 1 and 2 to both compute nodes 1 and 2. These nodes would be doing *redundant* work. However, notice that to compute the first two

gradients, we only need the output from one of these two compute nodes. We can also assign gradients 3 and 4 to compute nodes 3 and 4. A pictorial version is given in Figure 20 below.

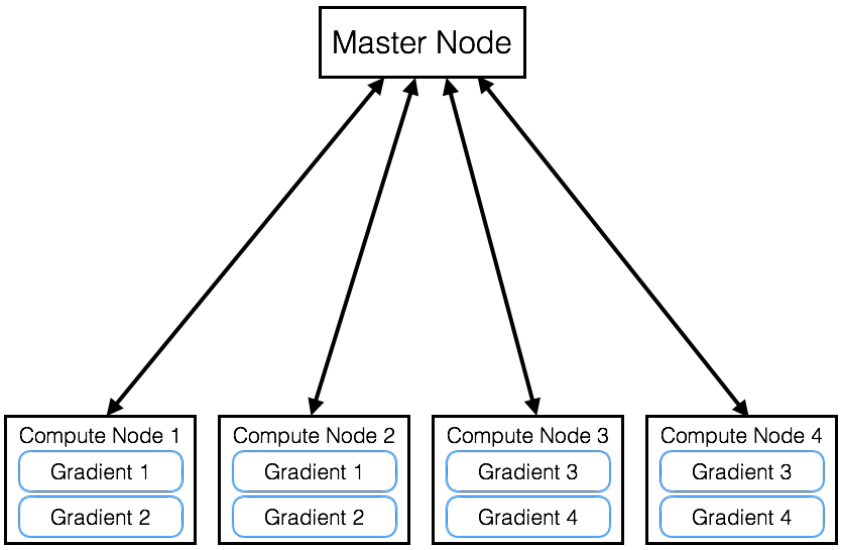


Figure 20: A master-worker system with 4 workers and 2 gradients per worker.

In general, one can verify that once any 3 of the nodes finish, we will have all the gradients. This means that we do not have to wait for the slowest node to finish. By doing some redundant computations, we have reduced the impact of stragglers. This comes at the expense of the number of tasks per worker, which is now 2 instead of 1. In general, there is a trade-off between redundancy, and the effect of stragglers. We would like to find ways of assigning tasks that balance the effect of stragglers and the number of tasks each node needs to compute.

There have also been many proposed asynchronous algorithms for distributed machine learning such as HOGWILD! [79]. In such algorithms, the machine learning model is updated each time any compute node finishes its task. In order to analyze the obtained model, most analyses of HOGWILD! assume a maximum amount of time that any

compute node takes to complete its tasks. Straggler nodes can make this assumption unrealistic in practical scenarios and can adversely affect the quality of the learned model. In both synchronous and asynchronous machine learning, we would like strategies to mitigate the effect of stragglers.

10.2 Previous Work

Several methods for mitigating the effect of stragglers have been recently proposed. These approaches include replicating jobs across redundant nodes and dropping stragglers in the case that the underlying computation is robust to errors [1, 83, 94]. The authors of [21] propose the use of redundant computations to avoid the effect of stragglers and improve the performance of traditional model training algorithms.

Recently, tools from coding theory have gained traction in an effort to mitigate stragglers. Lee et al. [54] proposed the use of techniques from coding theory to compensate for stragglers and communication bottlenecks in machine learning settings, especially in the computation of linear functions. Li et al. proposed using coding theory to reduce inter-server communication in the shuffling phase of MapReduce [56]. In [80], researchers proposed another coding-theoretic algorithm for speeding up distributed matrix multiplication in heterogeneous clusters. The use of codes for distributed matrix multiplication and linear operations on functions was also studied in [27], which analyzes the trade-off between the flexibility and sparsity of the code.

In [89], the authors propose *gradient coding*, a technique to exactly recover the sum of gradients from a subset of compute nodes. The authors show that we can recover k gradients from a subset of $k-s$ compute nodes, where each node computes $s+1$ gradients.

In other words, the algorithm is robust to s stragglers. Gradient coding is particularly relevant to synchronous distributed learning algorithms that involve computing sums of gradients, such as mini-batch stochastic gradient descent and full-batch gradient descent. While the gradient code construction in [89] was randomized, the authors in [78] use deterministic codes based on expander graphs to achieve similar results.

Most of the above results focus on exact reconstruction of a sum of functions. In many practical distributed settings, we may only require approximate reconstruction of the sum. In particular, parallel model training in machine learning settings has been shown to be robust to noise [64]. In some scenarios, noisy gradients may even improve the generalization performance of the trained model [67]. By only approximately reconstructing the desired function, we hope to increase the speed and tolerance to stragglers of our distributed algorithm. In fact, expander graphs, particularly Ramanujan graphs, can be used for such approximate reconstruction [78]. Unfortunately, expander graphs, especially Ramanujan graphs, can be expensive to compute in practice, especially for large numbers of compute nodes. Moreover, the desired parameters of the construction may be constrained according to underlying combinatorial rules.

10.2.1 Our Contributions

In this work, we use sparse graphs to create gradient codes capable of efficiently and accurately computing approximate gradients. More generally, these codes can be used to approximately recover any sum of functions. We formally introduce the approximate recovery problem and give a coding–theoretic interpretation of it. We also present and analyze two decoding techniques for approximate reconstruction, an optimal decoding

algorithm that has polynomial-time complexity, and an inexact decoding method that has linear complexity in the sparsity of the input.

We focus on two different codes that are efficiently computable and require only a logarithmic number of tasks per compute node. The first code is the Fractional Repetition Code (FRC) proposed in [89]. We show that FRCs can achieve small or zero error with high probability, even if a *constant fraction* of compute nodes are stragglers. However, we show that FRCs are susceptible to *adversarial stragglers*, where an adversary selects the worst-case set of stragglers. To get around this issue, we also present the Bernoulli Gradient Code (BGC) and the regularized Bernoulli Gradient Code (rBGC), whose constructions are based on sparse random graphs. We show that adversarial straggler selection in general codes is NP-hard, suggesting that these random code smay perform better against polynomial-time adversaries. We give explicit bounds on the error of BGCs and rBGCs that show that this potential tolerance to adversaries comes at the expense of a worse average-case error than FRCs. We provide simulations that support our theoretical results. These simulations show that there is a trade-off between the decoding complexity of a gradient code and its average- and worst-case performance.

10.3 Problem Statement

In the following, we will denote vectors and matrices in bold and scalars in standard script. For a vector $\mathbf{v} \in \mathbb{R}^m$, we will let $\|\mathbf{v}\|_2$ refer to its ℓ_2-norm. For any matrix $\mathbf{A}$, we will let $\mathbf{A}_{i,j}$ denote its (i, j) entry and let $\mathbf{a}_j$ denote its jth column. We will also let $\|\mathbf{A}\|_2$ denote its spectral norm while $\sigma_{\min}(\mathbf{A})$ will denote its smallest singular value. We will let $\mathbf{1}_m$ denote the $m \times 1$ all ones vector, while $\mathbf{1}_{n\times m}$ will denote the all ones $n \times m$

matrix. We define $\mathbf{0}_{n\times m}$ analogously.

In this work, we consider a distributed master-worker setup of n compute nodes, each of which is assigned with a maximum number of s tasks. The compute nodes can compute locally assigned tasks and they can send messages to the master node.

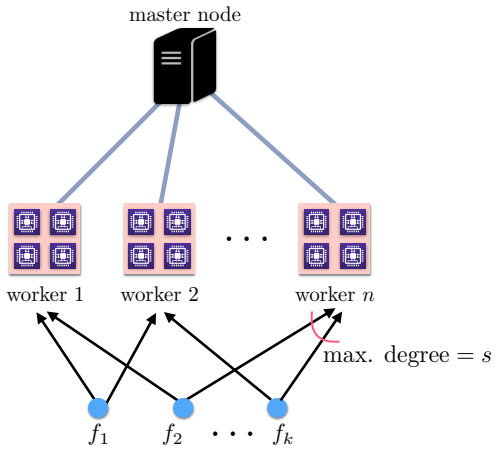


Figure 21: A master-worker architecture of distributed computation with multiple cores per machines. Each of the k functions is assigned to a subset of n compute nodes.

The goal of the master node is to compute the sum of k functions

$$f(\mathbf{x}) = \sum_{i=1}^{k} f_i(\mathbf{x}) = \mathbf{f}^T \mathbf{1_k} \tag{10.1}$$

in a distributed way, where $f_i : \mathbb{R}^d \to \mathbb{R}^w$, and any of the local functions f_i can be assigned to and computed locally by any of the n compute nodes. In order to keep local computation manageable, we assign at most s tasks to each compute node. Due to the straggler effect, we assume that we only get access to the output of $r < n$ non-straggler compute nodes. If we wish to exactly recover $f(\mathbf{x})$, then we need $r \geq k - s + 1$ [89]. In practice, we only need to approximately recover $f(\mathbf{x})$, which we may be able to do with many fewer non-stragglers.

The above setup is relevant to distributed learning algorithms, where we often wish to find some model $\mathbf{x}$ by minimizing

$$\ell(\mathbf{x}) = \sum_{i=1}^{k} \ell(\mathbf{x}; \mathbf{z}_i).$$

Here $\{\mathbf{z}_i\}_{i=1}^k$ are our training samples and $\ell(\mathbf{x}; \mathbf{z})$ is the loss function measuring the accuracy of the prediction served by model $\mathbf{x}$ with respect to data point $\mathbf{z}$. In order to find the model that minimizes the sum of losses $\ell(\mathbf{x})$, we often use first-order, or gradient based methods. For example, if we wanted to use gradient descent to find a good model, we would need to compute the full gradient of $\ell(\mathbf{x})$, given by

$$\nabla\ell(\mathbf{x}) = \sum_{i=1}^{k} \nabla\ell(\mathbf{x}; \mathbf{z}_i).$$

Letting $f_i(\mathbf{x}) = \nabla\ell(\mathbf{x}; \mathbf{z}_i)$, we arrive at the setup in 10.1. We focus our discussion on gradient descent for simplicity, but we note that the same setup applies for mini-batch SGD.

Our main question is whether we can develop gradient coding techniques that are robust to larger numbers of straggler nodes, if we allow for some error in the reconstruction of $\nabla\ell(x)$. We formally describe the approximate gradient coding problem below.

10.3.1 Approximate Gradient Coding

There are three components of an approximate gradient coding scheme.

1. Function assignments for each compute node.

2. The messages sent from each compute node to the master.

3. The decoding algorithm used by the master to recover an approximate sum of gradients.

First, each compute node is assigned with s functions to compute locally. In some version of the problem, we allow compute nodes to compute up to $\tilde{O}(s)$ functions, where $\tilde{O}$ hides poly-log factors.

After assigning tasks to each compute node, we let the compute nodes run local computations for some maximum amount of time. Afterwards, we may have compute nodes that either failed to compute some functions or are still running. These are the straggler nodes. During the approximate reconstruction of the sum, the master node will only use the output of the non-straggler nodes. We assume we only have access to the output of a subset of $r < n$ of the compute nodes. We want to use their output to compute the best approximation to $f(x)$ given by (10.1) possible. Here we assume that we can only linearly combine the results that the master received from the non-straggler nodes.

More formally, the task assignments are represented by a *function assignment matrix* $\mathbf{G}$, a $k \times n$ matrix where the support of column j indexes the functions assigned to compute node j. The entries of column j correspond to the coefficients of the linear combination of these local functions that the compute node sends back to the master once the compute node has completed its local computations.

Let $\mathbf{A}$ denote the $k \times r$ submatrix of $\mathbf{G}$ corresponding to the r non-straggler compute nodes. The minimum recovery error for a given subset matrix $\mathbf{A}$ is given by

$$\min_{\mathbf{x}} \ (\mathbf{f}^T \mathbf{A}\mathbf{x} - \mathbf{f}^T \mathbf{1_k})^2. \tag{10.2}$$

To better analyze this error, we define the *optimal decoding error* of a matrix $\mathbf{A}$.

Definition 10.1. *The optimal decoding error of a non-straggler matrix* $\mathbf{A}$ *is define as*

$$\mathrm{err}(\mathbf{A}) := \min_{\mathbf{x}} \|\mathbf{A}\mathbf{x} - \mathbf{1}_k\|_2^2.$$

The optimal decoding error quantifies how close $\mathbf{1}_k$ is to being in the span of the columns of $\mathbf{A}$. Note that if $\text{err}(\mathbf{A})$ is small, then the overall minimum recovery error is small, since

$$\min_{\mathbf{x}} \ (\mathbf{f}^T\mathbf{A}\mathbf{x} - \mathbf{f}^T\mathbf{1_k})^2 \leq \|\mathbf{f}\|_2^2 \, \text{err}(\mathbf{A}), \tag{10.3}$$

For a given matrix $\mathbf{A}$, let $\mathbf{A}^+$ denote its pseudo-inverse. Straightforward properties of the pseudo-inverse imply

$$\text{err}(\mathbf{A}) = \|\mathbf{A}\mathbf{A}^+\mathbf{1}_k - \mathbf{1}_k\|_2^2.$$

In general, we are interested in constructing function assignment matrices $\mathbf{G}$ such that submatrices $\mathbf{A}$ have small decoding error. Note that we can either consider the worst-case among all $\mathbf{A}$ or consider the setting where $\mathbf{A}$ is chosen uniformly at random. We will refer to these $\mathbf{G}$ as *approximate gradient codes*.

Approximate gradient codes were constructed in [78] using expander graphs. The authors show in particular that if $\mathbf{G}$ is the adjacency matrix of a Ramanujan graph, then the worst-case decoding error is relatively small. Unfortunately, such graphs may be expensive to compute in practice. To circumvent this issue, we use simplified random constructions.

10.4 Decoding

Once we receive the non-straggler matrix $\mathbf{A}$, we want to use it to approximate $\mathbf{1}_k$. We give two possible methods below. Because of the parallels to coding theory, we will refer to these as *decoding methods*. The first method will be referred to as *one-step decoding* because it is analogous to computing $\mathbf{u}_1$. The second will be referred to as *optimal decoding*.

Algorithm 5: One-step decoding algorithm.

Input : A $k \times r$ non-straggler matrix $\mathbf{A}$, $\rho > 0$.
Output: An approximation $\mathbf{v}$ to $\mathbf{1}_k$
1. Set $\mathbf{x} = \rho \mathbf{1}_r$.
2. Compute $\mathbf{v} = \mathbf{A}\mathbf{x}$.
3. Return $\mathbf{v}$.

Algorithm 6: Optimal decoding algorithm.

Input : A $k \times r$ non-straggler matrix $\mathbf{A}$.
Output: $\mathbf{v} = \text{argmin}_{\mathbf{w}} \|\mathbf{A}\mathbf{w} - \mathbf{1}_k\|_2^2$.
1. Find $\mathbf{A}^+$.
2. Compute $\mathbf{x} = \mathbf{A}^+ \mathbf{1}_k$.
3. Return $\mathbf{v} = \mathbf{A}\mathbf{x}$;

For the one-step decoding error, we will generally consider $\rho = \frac{k}{rs}$. If $\mathbf{G}$ has s entries in each column and row, then we would expect $\mathbf{A}$ to have roughly $\frac{rs}{k}$ entries in each row. If this holds exactly, then setting $\rho = \frac{k}{rs}$ will allow us to exactly reconstruct the gradient.

A decoding method analogous to the one-step decoding method was previously used in [78]. Note that the one-step decoding method is more efficient to compute than the optimal decoding, especially when $\mathbf{A}$ is ill-conditioned or k is large. Moreover, we can apply the one-step decoding method even if we do not have direct access to $\mathbf{A}$ but can compute matrix-vector product $\mathbf{A}\mathbf{x}$. The one-step decoding method allows us to avoid holding the entire matrix $\mathbf{A}$ in the memory of the master compute node in settings where this is not possible.

It is straightforward to see that if $\mathbf{v}$ is the optimal decoding of $\mathbf{A}$, then $\|\mathbf{v} - \mathbf{1}_k\|_2^2 = \text{err}(\mathbf{A})$. On the other hand, if $\mathbf{v}$ is the one-step decoding of $\mathbf{A}$ then $\|\mathbf{v} - \mathbf{1}_k\|_2^2 \geq \text{err}(\mathbf{A})$. We define the *one-step decoding error* of $\mathbf{A}$ as follows.

Definition 10.2. *For a given $\rho > 0$, the one-step error of $\mathbf{A}$ is defined by*

$$\mathrm{err}_1(\mathbf{A}) := \|\rho \mathbf{A}\mathbf{1}_r - \mathbf{1}_k\|_2^2.$$

10.5 Mathematical Perspective and Main Results

Algebraically, we want matrices $\mathbf{G} \in \mathbb{R}^{k \times k}$ such that for most (or all) column submatrices $\mathbf{A} \in \mathbb{R}^{k \times r}$, the ones vector $\mathbf{1}_k$ is approximately in the column span of $\mathbf{A}$. In general, this question becomes more difficult when we enforce a desired column-sparsity of $\mathbf{G}$. Given $\mathbf{A}$, let $\Pi_{\mathbf{A}}$ denote the projection on to the column span of $\mathbf{A}$. When applicable, we suppose that the columns of $\mathbf{A}$ are chosen through some random process. This suggests three primary questions concerning the geometry of column-sparse matrices.

Question 10.3. *How do we design a column-sparse matrix $\mathbf{G}$ so that*

$$\mathbb{E}_{\mathbf{A}} \|\Pi_{\mathbf{A}}(\mathbf{1}_k) - \mathbf{1}_k\|_2^2$$

is minimized?

Question 10.4. *How do we design a column-sparse matrix $\mathbf{G}$ so that*

$$\max_{\mathbf{A}} \|\Pi_{\mathbf{A}}(\mathbf{1}_k) - \mathbf{1}_k\|_2^2$$

is minimized?

Question 10.5. *Fix $\epsilon, \delta > 0$. For which s are there $\mathbf{G}$ with s non-zero entries in each column such that*

$$\mathbb{P}_{\mathbf{A}} \left(\|\Pi_{\mathbf{A}}(\mathbf{1}_k) - \mathbf{1}_k\|_2^2 \geq \epsilon \right) \leq \delta.$$

To make these results practically useful, we may want to specifically design $\mathbf{G}$ that are efficiently computable. Our main theorem shows that there is an efficiently computable code that has small or zero decoding error with high probability, even with a constant fraction of stragglers.

Theorem 10.6. *Suppose that $r = (1-\delta)k$. Then there is an efficiently computable $\mathbf{G}$ with*

$$s = \frac{\log(k)}{\log(\frac{1}{\delta})}$$

non-zero entries in each column such that if $\mathbf{A}$ is chosen uniformly at random,

$$\mathbb{P}_{\mathbf{A}}\left(\Pi_{\mathbf{A}}(\mathbf{1}_k) = \mathbf{1}_k\right) \geq 1 - \frac{1}{k}.$$

This is a simplified version of Theorem 11.5 below. Whether this $\mathbf{G}$ is optimal is unknown and an interesting open question. In machine learning terms, this result implies that for any k and a constant fraction of stragglers r, there is a code with sparsity $\Theta(\log(k))$ that can exactly reconstruct the gradient with high probability. While this code achieves smaller error for most $\mathbf{A}$ than previously designed codes, we show that it comes at the expense of the worst-case error, which can be $\Theta(k)$. Moreover, this worst-case be computed efficiently by an adversary.

On the other hand, we also show that in general, adversarial straggler selection is NP-hard. In order to counter polynomial-time adversaries, we give another approximate gradient code that utilizes randomness. We also bound the decoding error of this code. We show the following theorem, a simpler version of Theorem 11.24 below.

Theorem 10.7. *There is a randomized, efficiently computable $\mathbf{G}$ with at most $2s$ entries in each column such that if $\mathbf{A}$ is chosen uniformly at random, then with probability at*

least $1 - \frac{1}{k}$,

$$\frac{1}{k}\|\Pi_{\mathbf{A}}(\mathbf{1}_k) - \mathbf{1}_k\|_2^2 = O\left(\frac{1}{s}\right).$$

Chapter 11

Approximate Gradient Codes

Recipe 14: Challah with Raisins

Ingredients

- 3 $\frac{3}{4}$ teaspoons active dry yeast
- $\frac{1}{2}$ cup plus 1 tablespoon sugar
- 1 $\frac{3}{4}$ cups lukewarm water
- $\frac{1}{2}$ cup olive oil
- 5 eggs
- 1 tablespoon salt
- 8 $\frac{1}{2}$ cups flour
- $\frac{1}{2}$ cup raisins

Preparation

1. Place raisins in hot water and let them get plump. Drain well.
2. In a large bowl, dissolve yeast and 1 tablespoon sugar in lukewarm water. Set aside until foamy.
3. Whisk in oil, and 4 eggs, one at a time, followed by the remaining sugar and salt. Gradually add flour to form dough. Knead until smooth.
4. Turn dough on to floured surface. Clean and grease the bowl and return dough to bowl. Cover with plastic wrap and let it rise in a warm place for 1 hour. It should come close to doubling in size. Remove plastic wrap and punch down dough. Cover and let rise for another half-hour.

5. Knead raisins into challah and form loaves. Place loaves on greased cookie sheets. Beat remaining egg with 1 tablespoon of water and brush loaves. Let dough rise for another hour.

6. Heat oven to 375° F. Bake for 30 to 40 minutes or until golden-brown.

11.1 Fractional Repetition Codes

We would like to devise a code that achieves small error with high probability in the setting that our stragglers are chosen randomly. In fact, this can be achieved by the *fractional repetition code* (FRC) used in [89]. Note that [89] only considers this code for exact reconstruction of the gradient over all subsets of stragglers. This code can still be used when we only want approximately reconstruct the sum of k gradients with high probability.

This scheme works by replicating certain tasks between compute nodes. Suppose that we have k tasks and compute nodes and we want each compute node to compute s tasks. Without loss of generality, we suppose that s divides k. The assignment matrix $\mathbf{G}_{\text{frac}}$ for this scheme is then defined by

$$\mathbf{G}_{\text{frac}} = \begin{pmatrix} \mathbf{1}_{s\times s} & \mathbf{0}_{s\times s} & \mathbf{0}_{s\times s} & \dots & \mathbf{0}_{s\times s} \\ \mathbf{0}_{s\times s} & \mathbf{1}_{s\times s} & \mathbf{0}_{s\times s} & \dots & \\ \mathbf{0}_{s\times s} & \mathbf{0}_{s\times s} & \mathbf{1}_{s\times s} & \dots & \mathbf{0}_{s\times s} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{0}_{s\times s} & \mathbf{0}_{s\times s} & \mathbf{0}_{s\times s} & \dots & \mathbf{1}_{s\times s} \end{pmatrix}.$$

We assume that the $k \times r$ matrix $\mathbf{A}_{\text{frac}}$ of non-stragglers has columns that are sampled

uniformly without replacement from the k columns of $\mathbf{G}_{\text{frac}}$. We first compute the expected one-step decoding error. Let $\mathbf{a}_i$ denote column i of $\mathbf{A}_{\text{frac}}$.

Lemma 11.1.

$$\mathbb{E}[\mathbf{a}_i^T \mathbf{a}_j] = \begin{cases} s & , \ i = j \\ \dfrac{s^2}{k} - \dfrac{s}{k} & , \ i \neq j \end{cases}.$$

Proof. Fix $\mathbf{a}_i$. Since $\mathbf{a}_i$ has s non-zero entries that are all 1, $\mathbf{a}_i^T \mathbf{a}_i = s$. Next, suppose $j \neq i$. By the construction of $\mathbf{G}_{\text{frac}}$, there are only $s-1$ columns of $\mathbf{G}_{\text{frac}}$ that are not orthogonal to $\mathbf{a}_i$. Note that $\mathbf{a}_j$ is a duplicate of $\mathbf{a}_i$ with probability $\frac{s-1}{k}$. If this holds, then $\mathbf{a}_i^T \mathbf{a}_j = s$, and it is 0 if this does not hold. Therefore, for $i \neq j$,

$$\mathbb{E}[\mathbf{a}_i^T \mathbf{a}_j] = s\left(\frac{s-1}{k}\right) = \frac{s^2}{k} - \frac{s}{k}.$$

□

Theorem 11.2. *Setting $\rho = \frac{k}{rs}$ in the one-step decoding method, we have*

$$\mathbb{E}\left[\text{err}_1(\mathbf{A}_{frac})\right] = \frac{\delta k}{(1-\delta)s} - \frac{1}{1-\delta}\left(\frac{s-1}{s}\right).$$

Proof. Using linearity of expectation, we have

$$\begin{aligned} \mathbb{E}\left[\left\|\frac{k}{rs}\mathbf{A}_{\text{frac}}\mathbf{1}_r - \mathbf{1}_k\right\|_2^2\right] &= \mathbb{E}\left[\frac{k^2}{r^2s^2}\mathbf{1}_r^T\mathbf{A}_{\text{frac}}^T\mathbf{A}_{\text{frac}}\mathbf{1}_r - \frac{2k}{rs}\mathbf{1}_k^T\mathbf{A}_{\text{frac}}\mathbf{1}_r + \mathbf{1}_k^T\mathbf{1}_k\right] \\ &= \frac{k^2}{r^2s^2}\mathbf{1}_r^T\mathbb{E}[\mathbf{A}_{\text{frac}}^T\mathbf{A}_{\text{frac}}]\mathbf{1}_r - \frac{2k}{rs}s\mathbf{1}_r^T\mathbf{1}_r + k \\ &= \frac{k^2}{r^2s^2}\sum_{i,j}\mathbb{E}[\mathbf{a}_i^T\mathbf{a}_j] - k. \end{aligned}$$

Between step 1 and step 2 we used the fact that the columns of $\mathbf{A}_{\text{frac}}$ all have s

non-zero entries so $\mathbf{1}_k^T \mathbf{A}_{\text{frac}} = s\mathbf{1}_r$. Applying Lemma 11.1,

$$\begin{aligned}
\mathbb{E}\left[\left\|\frac{k}{rs}\mathbf{A}_{\text{frac}}\mathbf{1}_r - \mathbf{1}_k\right\|_2^2\right] &= \frac{k^2}{r^2s^2}\sum_{i,j}\mathbb{E}[\mathbf{a}_i^T\mathbf{a}_j] - k \\
&= \frac{k^2}{r^2s^2}\left(rs + r(r-1)\left(\frac{s^2}{k} - \frac{s}{k}\right)\right) - k \\
&= \frac{k^2}{r^2s^2}\left(rs + \frac{r^2s^2}{k} - \frac{r^2s}{k} - \frac{rs^2}{k} + \frac{rs}{k}\right) - k \\
&= \frac{k^2}{rs} - \frac{k}{s} - \frac{k}{r} + \frac{k}{rs} \\
&= \frac{k}{s}\left(\frac{k}{r} - 1\right) - \frac{k}{r} + \frac{k}{rs} \\
&= \frac{\delta k}{(1-\delta)s} - \frac{1}{1-\delta}\left(\frac{s-1}{s}\right).
\end{aligned}$$

□

Next, we consider the optimal decoding error of $\mathbf{A}_{\text{frac}}$. Note that each column of $\mathbf{A}_{\text{frac}}$ has k/s distinct possibilities,

$$\mathbf{v}_1 = \begin{pmatrix} \mathbf{1}_s \\ \mathbf{0}_s \\ \vdots \\ \mathbf{0}_s \end{pmatrix}, \mathbf{v}_2 = \begin{pmatrix} \mathbf{0}_s \\ \mathbf{1}_s \\ \vdots \\ \mathbf{0}_s \end{pmatrix}, \ldots, \mathbf{v}_{k/s} = \begin{pmatrix} \mathbf{0}_s \\ \mathbf{0}_s \\ \vdots \\ \mathbf{1}_s \end{pmatrix}.$$

There are s copies of each $\mathbf{v}_i$ in $\mathbf{G}_{\text{frac}}$ and $\mathbf{A}_{\text{frac}}$ has a set of columns given by sampling r of these without replacement. It is straightforward to see that $\text{err}(\mathbf{A}_{\text{frac}}) = \alpha s$, where α is the number of i such that $\mathbf{v}_i$ is not a column of $\mathbf{A}_{\text{frac}}$.

Let $Y_1, \ldots, Y_{k/s}$ denote the random variables where Y_i indicates whether every column from the ith block is sampled. Note that we then have

$$\text{err}(\mathbf{A}) = \sum_{i=1}^{k/s} sY_i. \tag{11.1}$$

Each Y_i is 1 iff each of the s columns in the ith block are sampled as part of the r. Therefore,

$$\mathbb{P}(Y_i = 1) = \frac{\binom{k-s}{r-s}}{\binom{k}{r}}. \tag{11.2}$$

Combining (11.1) and (11.2), we get the following theorem.

Theorem 11.3.

$$\mathbb{E}[\mathrm{err}(\mathbf{A}_{frac})] = k\frac{\binom{k-s}{r-s}}{\binom{k}{r}}.$$

We would now like high-probability bounds on $\mathrm{err}(\mathbf{A})$. By (11.1), this reduces to bounding how many of the Y_i are non-zero. This can be done via standard techniques concerning with-replacement sampling.

Theorem 11.4. *If $\mathbf{G}_{frac}$ is $k \times k$, for all $\alpha \in \mathbb{Z}_{\geq 0}$,*

$$\mathbb{P}\big(\,\mathrm{err}(\mathbf{A}_{frac}) \leq \alpha s\big) \geq 1 - \binom{k/s}{\alpha+1}\frac{\binom{k-(\alpha+1)s}{r}}{\binom{k}{r}}.$$

Proof. Fix $T \subseteq \{1, 2, \ldots, k/s\}$ such that $|T| = \alpha + 1$. Then

$$\mathbb{P}\big(\forall i \in T,\ \mathbf{v}_i \text{ is not a column of } \mathbf{A}_{\mathrm{frac}}\big) = \frac{\binom{k-(\alpha+1)s}{r}}{\binom{k}{r}}.$$

Taking a union bound, this implies

$$\bigcup_{T:|T|=\alpha+1} \mathbb{P}\big(\forall i \in T,\ \mathbf{v}_i \text{ is not a column of } \mathbf{A}_{\mathrm{frac}}\big) \leq \binom{k/s}{\alpha+1}\frac{\binom{k-(\alpha+1)s}{r}}{\binom{k}{r}}.$$

Note that the probability that we have no more than α of the $\mathbf{v}_i$ missing from the columns of $\mathbf{A}$ is the probability that $\mathrm{err}(\mathbf{A}_{\mathrm{frac}}) \leq \alpha s$. Therefore,

$$\mathbb{P}\big(\,\mathrm{err}(\mathbf{A}_{\mathrm{frac}}) \leq \alpha s\big) \geq 1 - \binom{k/s}{\alpha+1}\frac{\binom{k-(\alpha+1)s}{r}}{\binom{k}{r}}.$$

□

While this exact expression is complicated, this result easily shows that if $s = \Omega(\log(k))$, then with probability at least $1 - \frac{1}{k}$, $\operatorname{err}(\mathbf{A}) = O(\log(k))$. Recall that the number of non-stragglers $r = (1-\delta)k$ for $\delta \in (0,1)$.

Theorem 11.5. *Suppose*

$$s \geq \left(1 + \frac{1}{1+\alpha}\right) \frac{\log(k)}{1-\delta}.$$

Then

$$\mathbb{P}\big(\operatorname{err}(\mathbf{A}_{frac}) > \alpha s\big) \leq \frac{1}{k}.$$

Proof. By Theorem 11.4, we have

$$\mathbb{P}\big(\operatorname{err}(\mathbf{A}_{\text{frac}}) > \alpha s\big) \leq \binom{k/s}{\alpha+1} \frac{\binom{k-(\alpha+1)s}{r}}{\binom{k}{r}}.$$

Simple estimates show

$$\begin{aligned}\frac{\binom{k-(\alpha+1)s}{r}}{\binom{k}{r}} &= \frac{(k-(\alpha+1)s)(k-(\alpha+1)s-1)\dots(k-(\alpha+1)s-r+1)}{k(k-1)\dots(k-r+1)} \\ &\leq \left(\frac{(k-(\alpha+1)s)}{k}\right)^r.\end{aligned}$$

Therefore,

$$\mathbb{P}\big(\operatorname{err}(\mathbf{A}_{\text{frac}}) > \alpha s\big) \leq k^{\alpha+1} \left(\frac{(k-(\alpha+1)s)}{k}\right)^r.$$

We wish to show that for $s \geq (1+\frac{1}{1+\alpha})\log(k)/(1-\delta)$, the right-hand side of this equation is at most $\frac{1}{k}$. Manipulating, this is equivalent to s satisfying

$$\frac{(\alpha+1)s}{k} \geq \left(1 - k^{-(\alpha+2)/r}\right). \tag{11.3}$$

Since $s \geq (1+\frac{1}{1+\alpha})\log(k)/(1-\delta)$, we have

$$\frac{(\alpha+1)s}{k} \geq \frac{(\alpha+2)\log(k)}{r}. \tag{11.4}$$

Letting $\beta = (\alpha + 2)\log(k)/r$, (11.4) implies that (11.3) holds if

$$\beta \geq 1 - e^{-\beta}.$$

Since this occurs for all $\beta \geq 0$, the desired result is shown. □

Corollary 11.6. *Suppose*

$$s \geq \frac{2\log(k)}{1-\delta}.$$

Then

$$\mathbb{P}\big(\operatorname{err}(\mathbf{A}_{frac}) > 0\big) \leq \frac{1}{k}.$$

Note that this implies Theorem 10.6. While FRCs have demonstrably small optimal decoding error when the stragglers are selected randomly, we will later show that it does not perform well when the stragglers are selected adversarially. In order to improve our tolerance to adversarial stragglers, we will develop a coding scheme based on random graphs.

11.2 Adversarial Stragglers

11.2.1 Adversarial Stragglers and Fractional Repetition Codes

While FRCs have small average-case error, their worst case-error is large. Worse, it is computationally efficient to find these worst-case straggler sets. In fact, they can be found in linear time in the number of compute nodes. Recall that the assignment matrix

$\mathbf{G}_{\text{frac}}$ for FRC is defined by

$$\mathbf{G}_{\text{frac}} = \begin{pmatrix} \mathbf{1}_{s\times s} & \mathbf{0}_{s\times s} & \mathbf{0}_{s\times s} & \dots & \mathbf{0}_{s\times s} \\ \mathbf{0}_{s\times s} & \mathbf{1}_{s\times s} & \mathbf{0}_{s\times s} & \dots & \\ \mathbf{0}_{s\times s} & \mathbf{0}_{s\times s} & \mathbf{1}_{s\times s} & \dots & \mathbf{0}_{s\times s} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{0}_{s\times s} & \mathbf{0}_{s\times s} & \mathbf{0}_{s\times s} & \dots & \mathbf{1}_{s\times s} \end{pmatrix}.$$

As previously noted, each column of $\mathbf{A}_{\text{frac}}$ has k/s distinct possibilities,

$$\mathbf{v}_1 = \begin{pmatrix} \mathbf{1}_s \\ \mathbf{0}_s \\ \vdots \\ \mathbf{0}_s \end{pmatrix}, \mathbf{v}_2 = \begin{pmatrix} \mathbf{0}_s \\ \mathbf{1}_s \\ \vdots \\ \mathbf{0}_s \end{pmatrix}, \dots, \mathbf{v}_{k/s} = \begin{pmatrix} \mathbf{0}_s \\ \mathbf{0}_s \\ \vdots \\ \mathbf{1}_s \end{pmatrix}.$$

There are s copies of each $\mathbf{v}_i$ in $\mathbf{G}_{\text{frac}}$. As long as one instance of $\mathbf{v}_i$ is a non-straggler, then we will recover these s entries in $\mathbf{1}_k$. If all s are stragglers, then our optimal decoding vector $\tilde{\mathbf{x}} = \text{argmin}_{\mathbf{x}} \|\mathbf{A}\mathbf{x} - \mathbf{1}_k\|_2^2$ will have zeros in the corresponding s entries. This contributes s to the optimal decoding error.

One can therefore see that if we wish to pick r non-stragglers adversarially, then we would pick all of the s copies of each $\mathbf{v}_i$ above, for a total of r/s blocks. Here, we assume that s divides r for simplicity. Without loss of generality, we can simply select the first r columns of $\mathbf{G}_{\text{frac}}$ to be non-stragglers. If $\mathbf{G}_{\text{frac}}$ has its columns permuted then we can simply select all columns corresponding to the first r/s blocks. There are then $(k-r)/s$ blocks missing from $\mathbf{A}$. Each contributes s to the optimal decoding error.

Furthermore, the optimal decoding error increases by s if and only if all of the stragglers corresponding to one of the k/s blocks are all stragglers. Therefore, the adversarial

selection described above gives the worst-case error for this scheme. Moreover, the adversary can find this set in $O(k)$ operations if they have full-knowledge of what coding scheme is being used. Even if they do not have this knowledge, an adversary can check for this coding scheme and find the worst-case straggler set in $O(sk)$ operations if they only have access to the matrix $\mathbf{G}$. This implies the following theorem.

Theorem 11.7. *Suppose that the non-stragglers are selected adversarially. In the worst-case, the non-straggler matrix* $\mathbf{A}$ *will satisfy*

$$\text{err}(\mathbf{A}) = k - r.$$

Moreover, worst-case straggler sets can be found in polynomial time.

Note that assuming that $r = (1 - \delta)k$ for some constant δ, then the adversarial optimal decoding error is $\Theta(k)$. This is in stark contrast to Theorem 11.5, which showed that if the stragglers are selected randomly and $s = \Omega(\log k)$, then with high probability $\text{err}(\mathbf{A}) = O(\log k)$. Also note that since $\text{err}_1(\mathbf{A}) \geq \text{err}(\mathbf{A})$, we derive the same adversarial result for the one-step decoding error.

11.2.2 Adversarial Straggler Selection is NP-hard

In this section, we discuss what happens when the stragglers are chosen *adversarially* instead of randomly. We show that adversarial straggler selection is NP-hard. This demonstrates that in general, adversaries with polynomial-time computations cannot find the set of stragglers that maximizes the decoding error. In such cases, the average-case error may be a more useful indicator of how well a gradient code performs.

To show that general adversarial selection is NP-hard, we first define two problems.

Definition 11.8 (Densest k-Subgraph Problem)**.** *The k-densest subgraph problem (DkS) asks, given a graph (V, E), what k-vertex subgraph contains the most edges.*

As shown in [3], this problem is NP-hard, even if we restrict to regular graphs. We now formally define the adversarial straggler problem.

Definition 11.9 (Adversarial Straggler Problem)**.** *Fix a constant $\rho > 0$. The r-adversarial straggler problem (r-ASP) asks, given a square matrix $\mathbf{G} \in \mathbb{R}^{n\times n}$, which column-submatrix $\mathbf{A}$ maximizes*

$$\|\rho \mathbf{A}\mathbf{1}_r - \mathbf{1}_n\|_2^2.$$

Note that this is the form that one-step decoding takes. We will show that for any $\rho \in (0, \frac{2}{3})$, this problem is NP-hard, even when we restrict to $\mathbf{G} \in \{0,1\}^{k\times k}$ with at most s non-zero entries in each column.

Theorem 11.10. *For any $\rho \in (0, \frac{2}{3})$, the adversarial straggler problem is NP-hard. This holds even if we restrict to matrices $\mathbf{G} \in \mathbb{R}^{k\times k}$ with entries in $\{0, 1\}$ and at most $s \geq 2$ non-zero entries per column.*

In fact, the proof of our theorem also shows that if we instead consider all matrices $\mathbf{G} \in \mathbb{R}^{k\times n}$ where $k \geq n$, then r-ASP is NP-hard for any $\rho > 0$.

Proof. We will give a reduction from DkS on d-regular graphs to r-ASP where $\mathbf{G}$ is boolean with at most d non-zero entries per column.

Let (V, E) be a d-regular graph on n vertices. Note that $|E| = nd$. We want to solve DkS for (V, E). Let $\mathbf{M}$ denote the adjacency matrix of (V, E). Note that DkS is equivalent to

$$\max_{\mathbf{x}} \mathbf{x}^T\mathbf{M}\mathbf{x} \ \ s.t.\ \mathbf{x} \in \{0,1\}^n,\ \|\mathbf{x}\|_0 = k.$$

Let $\mathbf{B}$ denote the unsigned incidence matrix of (V, E). That is, $\mathbf{B}$ is a $|E|\times|V|$ boolean matrix where the row corresponding to edge e has a 1 in column v iff e is incident to v. We will let $\mathbf{C}$ denote the $|E| \times |E|$ matrix given by adding $|E| - |V| = n(d-1)$ zero columns to $\mathbf{B}$. Note that $\mathbf{C}$ is a square $nd \times nd$ boolean matrix with at most d non-zero entries in each column since (V, E) is d-regular.

Let $r = k + (n-1)d$. Consider r-ASP on $\mathbf{C}$. This is equivalent to finding a vector $\mathbf{x} \in \{0,1\}^{|E|}$ with $\|\mathbf{x}\|_0 = r$ that maximizes

$$\|\rho\mathbf{C}\mathbf{x} - \mathbf{1}_{nd}\|_2^2.$$

Straightforward computations show

$$\begin{aligned}\|\rho\mathbf{C}\mathbf{x} - \mathbf{1}_{nd}\|_2^2 &= \rho^2\mathbf{x}^T\mathbf{C}^T\mathbf{C}\mathbf{x} - 2\rho\mathbf{1}_{nd}^T\mathbf{C}\mathbf{x} + \mathbf{1}_{nd}^T\mathbf{1}_{nd} \\ &= \rho^2\mathbf{x}^T\mathbf{C}^T\mathbf{C}\mathbf{x} - 2\rho[d\mathbf{1}_n^T \;\; \mathbf{0}_{n(d-1)}^T]\mathbf{x} + nd.\end{aligned}$$

Let $\mathbf{x} = \begin{pmatrix}\mathbf{y}\\ \mathbf{z}\end{pmatrix}$ where $\mathbf{y} \in \{0,1\}^n, \mathbf{z} \in \{0,1\}^{n(d-1)}$. Note that $\mathbf{y}$ corresponds to which of the columns of $\mathbf{B}$ we select, while $\mathbf{z}$ corresponds to columns of $\mathbf{0}$ we select. Recall that $\|\mathbf{y}\|_0 + \|\mathbf{z}\|_0 = r = t + n(d-1)$.

Note that $\mathbf{B}^T\mathbf{B} = \mathbf{M} + \mathbf{I}d$. This implies

$$\mathbf{C}^T\mathbf{C} = \begin{pmatrix}\mathbf{M} + \mathbf{I}d & \mathbf{0}\\ \mathbf{0} & \mathbf{0}\end{pmatrix}.$$

We then get

$$\begin{aligned}\|\rho\mathbf{C}\mathbf{x} - \mathbf{1}_{nd}\|_2^2 &= \rho^2\mathbf{x}^T\mathbf{C}^T\mathbf{C}\mathbf{x} - 2\rho[d\mathbf{1}_n^T \;\; \mathbf{0}_{n(d-1)}^T]\mathbf{x} + nd \\ &= \rho^2\mathbf{y}^T\mathbf{M}\mathbf{y} + d\rho^2\mathbf{y}^T\mathbf{y} - 2\rho d\mathbf{1}_n^T\mathbf{y} + nd \\ &= \rho^2\mathbf{y}^T\mathbf{M}\mathbf{y} + d\rho^2\|\mathbf{y}\|_0 - 2\rho d\|\mathbf{y}\|_0 + nd.\end{aligned}$$

Therefore, r-ASP in this setting is equivalent to maximizing, over $\mathbf{y} \in \{0,1\}^n, \mathbf{z} \in \{0,1\}^{n(d-1)}$ such that $\|\mathbf{y}\|_0 + \|\mathbf{z}\|_0 = r = k + n(d-1)$, the quantity

$$\rho^2 \mathbf{y}^T \mathbf{M} \mathbf{y} + d\rho^2 \|\mathbf{y}\|_0 - 2\rho d \|\mathbf{y}\|_0 + nd. \tag{11.5}$$

Define

$$f(\mathbf{y}) = \rho^2 \mathbf{y}^T \mathbf{M} \mathbf{y} + (\rho^2 - 2\rho) d \|\mathbf{y}\|_0.$$

Note that if we fix a binary $\mathbf{y}$ such that $\|\mathbf{y}\|_0 = a$, then

$$f(\mathbf{y}) = \rho^2 \mathbf{y}^T \mathbf{M} \mathbf{y} + (\rho^2 - 2\rho) da.$$

Therefore, maximizing this quantity corresponds to finding the a-densest subgraph of (V, E). We now must show that when we maximize this over $\mathbf{y}$ and $\mathbf{z}$, then the solution will always have $\|\mathbf{y}\|_0 = k$. Since $r = k + n(d-1)$, it suffices to show that $\mathbf{y}$ is as sparse as possible.

To show that this is the case, we will show that for $\rho \in (0, \frac{2}{3})$, increasing the sparsity of $\mathbf{y}$ by 1 will only decrease the objective function. Suppose that $\|\mathbf{y}\|_0 = a$ and it has support S. Say that $\mathbf{y}'$ satisfies $\|\mathbf{y}'\|_0 = a + 1$ and it has support S' where $S \subseteq S'$. Let T, T' denote the vertex subgraphs of (V, E) corresponding to S, S', and let $e(S), e(S')$ denote the number of edges in these subgraphs. Note that $\mathbf{y}^T \mathbf{M} \mathbf{y} = 2e(S)$. We then have

$$f(\mathbf{y}) = 2\rho^2 e(S) + (\rho^2 - 2\rho) da.$$

$$f(\mathbf{y}') = 2\rho^2 e(S') + (\rho^2 - 2\rho) d(a+1).$$

Note that since (V, E) is d-regular, $e(S') \leq e(S) + d$. Therefore,

$$\begin{aligned} f(\mathbf{y}') - f(\mathbf{y}) &= 2\rho^2(e(S') - e(S)) + (\rho^2 - 2\rho)d \\ &\leq 2\rho^2 d + \rho^2 d - 2\rho d \\ &= 3\rho^2 d - 2\rho d. \end{aligned}$$

For $\rho \in (0, \frac{2}{3})$, this quantity is negative. Therefore, increasing the sparsity of $\mathbf{y}$ will decrease the objective function $f(\mathbf{y})$. Therefore, the maximum of the r-ASP problem applied to $\mathbf{C}$ will have $\mathbf{y}$ as sparse as possible. Since $r = k+n(d-1)$ and $\|\mathbf{z}\|_0 \leq n(d-1)$, this implies that the maximum occurs at $\|\mathbf{y}\|_0 = k$. Let S denote the support of $\mathbf{y}$. The objective function in (11.5) is then equal to

$$2\rho^2 e(S) + d\rho^2 t - 2\rho dt + nd. \tag{11.6}$$

This is clearly maximized when S is the set of vertices forming the densest k-subgraph. □

11.3 Bernoulli Gradient Codes

In this section we will consider the case that $\mathbf{G}$ has entries that are Bernoulli random variables. For a given s, k, we will refer to the Bernoulli coding scheme as setting, for $i \in \{1, \ldots, k\}, j \in \{1, \ldots, n\}$, $\mathbf{G}_{i,j} = \text{Bernoulli}(s/k)$. Intuitively, by injecting randomness into the construction of $\mathbf{G}$, we improve our tolerance to adversarial stragglers. This comes as the cost of worse average-case error. While [78] shows that if $\mathbf{G}$ is a Ramanujan graph then we have strong bounds on its adversarial decoding error, such graphs are notoriously tricky to compute. By using Bernoulli coding, we sacrifice a small amount of error in order to achieve a much simpler, efficiently computable coding scheme.

Suppose that the stragglers are selected uniformly at random. Then, the non-straggler submatrix $\mathbf{A}$ also has Bernoulli random entries. Note that the expected number of tasks computed by any compute node is s. This construction will allow us to derive high-probability bounds on the decoding error for $s > \log(k)$. We will later show that we can enforce the desired sparsity s of each column and maintain the same error. Moreover, enforcing this desired sparsity will let us extend these error bounds to the setting where $s < \log(k)$. In order to get a handle on the decoding error, we first develop a method to bound the optimal and one-step decoding errors.

11.3.1 Bounding the Decoding Error

Suppose we have a function assignment matrix $\mathbf{G}$ such that the sparsity of each column is exactly or approximately bounded by s. After performing the local computations on each compute node, we have access to a $k \times r$ submatrix $\mathbf{A}$ of the r non-stragglers. We assume that $r = (1 - \delta)k$ for some $\delta \geq 0$.

We would like to derive high probability bounds on $\text{err}(\mathbf{A})$ in order to bound the optimal decoding error of $\mathbf{A}$, as in (10.3). Unfortunately, it is not straightforward to directly bound this error for a random matrix $\mathbf{A}$ since it involves the pseudo-inverse of $\mathbf{A}$. Instead, we will use an algorithmic approach to bound the optimal decoding error with high probability. The following lemma is adapted from [107].

Lemma 11.11. *Let* $\mathbf{u}_0 = \mathbf{1}_k$*, and define*

$$\mathbf{u}_t = \mathbf{u}_{t-1} - \frac{\mathbf{A}\mathbf{A}^T}{\nu}\mathbf{u}_{t-1}.$$

If $\nu \geq \|\mathbf{A}\|_2^2$ *then*

$$\lim_{t\to\infty} \|\mathbf{u}_t\|_2^2 = \text{err}(\mathbf{A}).$$

Moreover, for all t, $\|\mathbf{u}_t\|_2^2 \geq \text{err}(\mathbf{A})$.

We refer to the $\mathbf{u}_t$ as the *algorithmic decoding error* of $\mathbf{A}$. To prove Lemma 11.11, we will use the following lemma, adapted from [107].

Lemma 11.12. *If $\mathbf{u}$ is in the column span of $\mathbf{A}$ and $\nu \geq \|\mathbf{A}\|_2^2$ then*

$$\left\| \left(\mathbf{I} - \frac{\mathbf{A}\mathbf{A}^T}{\nu} \right) \mathbf{u} \right\|_2 \leq \left(1 - \frac{\sigma_{\min}(\mathbf{A})^2}{\nu} \right) \|\mathbf{u}\|_2.$$

Proof of Lemma 11.11. Fix some $t \geq 1$. We can decompose $\mathbf{1}_k$ as $\mathbf{v} + \mathbf{w}$ where $\mathbf{v}$ is the orthogonal projection of $\mathbf{1}_k$ on to the column span of $\mathbf{A}$ and $\mathbf{w}$ is in the nullspace of $\mathbf{A}^T$. Note that this implies that $(\mathbf{I} - \mathbf{A}\mathbf{A}^T/\nu)\mathbf{w} = \mathbf{w}$. Therefore,

$$\mathbf{u}_t = \left(\mathbf{I} - \frac{\mathbf{A}\mathbf{A}^T}{\nu} \right)^t (\mathbf{v} + \mathbf{w}) = \left(\mathbf{I} - \frac{\mathbf{A}\mathbf{A}^T}{\nu} \right)^t \mathbf{v} + \mathbf{w}.$$

Since $\mathbf{v}$ is in the span of $\mathbf{A}$, $\left(\mathbf{I} - \frac{\mathbf{A}\mathbf{A}^T}{\nu} \right)^t \mathbf{v}$ is also in the span of $\mathbf{A}$ and orthogonal to $\mathbf{w}$. By Lemma 11.12,

$$\begin{aligned} \|\mathbf{u}_t\|_2^2 &= \left\| \left(\mathbf{I} - \frac{\mathbf{A}\mathbf{A}^T}{\nu} \right)^t \mathbf{v} \right\|_2 + \|\mathbf{w}\|_2^2. \\ &\leq \left(1 - \frac{\sigma_{\min}(\mathbf{A})^2}{\nu} \right)^{2t} \|\mathbf{v}\|_2^2 + \|\mathbf{w}\|_2^2. \end{aligned}$$

By construction, $\|\mathbf{w}\|_2^2 = \min_{\mathbf{x}} \|\mathbf{A}\mathbf{x} - \mathbf{1}_k\|_2^2$, completing the proof. □

Note that the $\mathbf{u}_t$ are defined as the iterates of projected gradient descent. Consider the setting where $\nu = \|\mathbf{A}\|_2^2$. The matrix $\mathbf{P} = \mathbf{A}\mathbf{A}^T/\|\mathbf{A}\|_2^2$ is a projection operator (ie. $\mathbf{P}^2 = \mathbf{P}$), and it projects a vector on to the column-span of $\mathbf{A}$. By letting $\mathbf{u}_t = \mathbf{u}_{t-1} - \mathbf{P}\mathbf{u}_{t-1}$, Lemma 11.11 one can show that this eventually converges to $\mathbf{u}_0 - \mathbf{A}\mathbf{A}^+\mathbf{u}_0$. In other words, we eventually converge to the component of $\mathbf{u}_0$ that is orthogonal to the range of $\mathbf{A}$. Taking $\mathbf{u}_0 = \mathbf{1}_k$, this eventually converges to $\text{err}(\mathbf{A})$.

We can better understand $\mathbf{u}_t$ by taking a combinatorial view. Note that $\mathbf{A}$ encodes a bipartite graph with k left vertices and r right vertices, where $\mathbf{A}_{ij}$ is 1 iff there is an edge between vertex i on the left and vertex j on the right. Column j of $\mathbf{A}$ corresponds to the incidence of the jth right vertex. In particular, the degree of vertex j on the right equal the number of tasks computed by compute node j. We can compute $\|\mathbf{u}_t\|_2^2$ in terms of walks on this bipartite graph.

Lemma 11.13. $\mathbf{1}_k^T(\mathbf{A}\mathbf{A}^T)^t\mathbf{1}_k$ *equals the number of paths of length* $2t$ *from a left vertex to a right vertex.*

Proof. Note that $(\mathbf{A}\mathbf{A}^T)_{ij}$ is the number of paths of length 2 from the vertex i to vertex j, where i, j are both left vertices. More generally, $(\mathbf{A}\mathbf{A}^T)_{ij}^t$ counts the weighted number of paths of length $2t$ from vertex i to vertex j. Therefore, $\mathbf{1}_k(\mathbf{A}\mathbf{A}^T)^t\mathbf{1}_k$ is the weighted number of paths of length $2t$ from a left vertex to a left vertex. □

Lemma 11.14. *Let* a_t *denote the weighted number of walks in the associated bipartite graph of* $\mathbf{A}$ *of length* $2t$ *starting and ending at a left vertex. Then*

$$\|\mathbf{u}_t\|_2^2 = a_0 - \binom{2t}{1}\frac{a_1}{\nu} + \binom{2t}{2}\frac{a_2}{\nu^2} - \ldots + \binom{2t}{2t}\frac{a_{2t}}{\nu^{2t}}.$$

Proof. By direct computation we have

$$\begin{aligned}
\|\mathbf{u}_t\|_2^2 &= \left\|\left(\mathbf{I} - \frac{\mathbf{A}\mathbf{A}^T}{\nu}\right)\mathbf{u}_{t-1}\right\|_2^2 \\
&= \left\|\left(\mathbf{I} - \frac{\mathbf{A}\mathbf{A}^T}{\nu}\right)^t\mathbf{1}_k\right\|_2^2 \\
&= \mathbf{1}_k^T\mathbf{1}_k - \binom{2t}{1}\frac{\mathbf{1}_k^T\mathbf{A}\mathbf{A}^T\mathbf{1}_k}{\nu} + \binom{2t}{2}\frac{\mathbf{1}_k^T(\mathbf{A}\mathbf{A}^T)^2\mathbf{1}_k}{\nu^2} - \ldots + \binom{2t}{2t}\frac{\mathbf{1}_k^T(\mathbf{A}\mathbf{A}^T)^{2t}\mathbf{1}_k}{\nu^{2t}}.
\end{aligned}$$

By Lemma 11.13, the result follows. □

While $\mathbf{u}_t$ may be difficult to bound for sufficiently large t, we can handle $\mathbf{u}_1$ more directly. Moreover, as theory and simulations will show, even $\mathbf{u}_1$ will give us good bounds on $\text{err}(\mathbf{A})$.

11.3.2 One-step Error of Bernoulli Gradient Codes

Recall that our function assignment matrix $\mathbf{G}$ has entries that are Bernoulli random variables with probability s/k. The non-straggler matrix $\mathbf{A}$ is a column submatrix and therefore also has Bernoulli random entries. We will view $\mathbf{A}$ as encoding a bipartite graph with k left vertices and r right vertices. We say that there is an edge between left vertex i and right vertex j iff $\mathbf{A}_{i,j} = 1$. Note that $\mathbb{E}[\mathbf{A}] = \frac{s}{k}\mathbf{1}_{k\times r}$. Therefore, the expected degree of any vertex in the associated bipartite graph is at most s. We will bound $\|\mathbf{A} - \mathbb{E}\mathbf{A}\|_2$ for various s and use this to bound $\text{err}(\mathbf{A})$. We will use the following lemma.

Lemma 11.15. *Suppose* $\|\mathbf{A} - \mathbb{E}\mathbf{A}\|_2 \leq \gamma$. *If* $\rho = \frac{k}{rs}$ *then*

$$\text{err}_1(\mathbf{A}) \leq \frac{\gamma^2 k}{(1-\delta)s^2}.$$

Proof. By standard norm properties,

$$\begin{aligned}
\left\| \frac{k}{rs}\mathbf{A}\mathbf{1}_r - \mathbf{1}_k \right\|_2^2 &= \left\| \frac{k}{rs}\mathbf{A}\mathbf{1}_r - \frac{k}{rs}\mathbb{E}\mathbf{A}\mathbf{1}_r \right\|_2^2 \\
&\leq \frac{k^2}{r^2 s^2}\|\mathbf{A} - \mathbb{E}\mathbf{A}\|_2^2 \|\mathbf{1}_r\|_2^2 \\
&\leq \frac{\gamma^2 k^2}{rs^2} \\
&= \frac{\gamma^2 k}{(1-\delta)s^2}.
\end{aligned}$$

□

This approach is analogous to bounding $\mathbf{u}_1$, as the following lemma shows.

Lemma 11.16. *Suppose that* $\|\mathbf{A} - \mathbb{E}\mathbf{A}\|_2 \leq \gamma$ *where* $\gamma \leq \sqrt{(1-\delta)}s$. *Then for* $\nu = \frac{rs^2}{k}$,

$$\mathbf{u}_1 \leq \frac{5\gamma^2 k}{(1-\delta)s^2}.$$

Proof. Recall that for a given ν, $\mathbf{u}_1$ is given by

$$\mathbf{u}_1 = \left\| \left(\mathbf{I} - \frac{\mathbf{A}\mathbf{A}^T}{\nu} \right) \mathbf{1}_k \right\|_2^2.$$

We then have

$$\begin{aligned}
\mathbf{u}_1 &= \left\| \mathbf{1}_k - \frac{\mathbf{A}\mathbf{A}^T}{\nu}\mathbf{1}_k \right\|_2^2 \\
&= \left\| \mathbf{1}_k - \frac{\mathbf{A}(\mathbb{E}\mathbf{A}^T + \mathbf{A}^T - \mathbb{E}\mathbf{A}^T)}{\nu}\mathbf{1}_k \right\|_2^2 \\
&\leq \left\| \mathbf{1}_k - \frac{\mathbf{A}\mathbb{E}\mathbf{A}^T}{\nu}\mathbf{1}_k \right\|_2^2 + \left\| \frac{\mathbf{A}(\mathbf{A}^T - \mathbb{E}\mathbf{A}^T)}{\nu}\mathbf{1}_k \right\|_2^2.
\end{aligned}$$

Note that $\mathbb{E}\mathbf{A} = \frac{s}{k}\mathbf{1}_{k\times r}$. Therefore, $\mathbf{1}_k = \frac{k}{rs}\mathbb{E}\mathbf{A}\mathbf{1}_r$. Using this fact and taking $\nu = \frac{rs^2}{k}$, we get

$$\begin{aligned}
\mathbf{u}_1 &\leq \left\| \mathbf{1}_k - \frac{\mathbf{A}\mathbb{E}\mathbf{A}^T}{\nu}\mathbf{1}_k \right\|_2^2 + \left\| \frac{\mathbf{A}(\mathbf{A}^T - \mathbb{E}\mathbf{A}^T)}{\nu}\mathbf{1}_k \right\|^2 \\
&\leq \left\| \frac{k}{rs}\mathbb{E}\mathbf{A}\mathbf{1}_r - \frac{s\mathbf{A}}{\nu}\mathbf{1}_r \right\|_2^2 + \frac{1}{\nu^2}\|\mathbf{A}\|_2^2\|\mathbf{A} - \mathbb{E}\mathbf{A}\|_2^2\|\mathbf{1}_k\|_2^2 \\
&\leq \frac{k^2}{rs^2}\|\mathbf{A} - \mathbb{E}\mathbf{A}\|_2^2 + \frac{k^3}{r^2s^4}\|\mathbf{A}\|_2^2\|\mathbf{A} - \mathbb{E}\mathbf{A}\|_2^2.
\end{aligned}$$

Note that since $\mathbb{E}\mathbf{A} = \frac{s}{k}\mathbf{1}_{k\times r}$, we have

$$\|\mathbb{E}\mathbf{A}\|_2 = \frac{s}{k}\|\mathbf{1}_{k\times r}\|_2 = \frac{s}{k}\sqrt{kr} = \sqrt{1-\delta}s.$$

Using our assumption that $\|\mathbf{A} - \mathbb{E}\mathbf{A}\|_2 \leq \gamma \leq \sqrt{(1-\delta)}s$, we find

$$\begin{aligned}\|\mathbf{A}\|_2^2 &\leq (\|\mathbb{E}\mathbf{A}\|_2 + \gamma)^2 \\ &\leq (2\sqrt{(1-\delta)}s)^2 \\ &\leq 4(1-\delta)s^2.\end{aligned}$$

Finally, this implies

$$\begin{aligned}\mathbf{u}_1 &\leq \frac{k^2}{rs^2}\|\mathbf{A} - \mathbb{E}\mathbf{A}\|_2^2 + \frac{k^3}{r^2s^4}\|\mathbf{A}\|_2^2\|\mathbf{A} - \mathbb{E}\mathbf{A}\|_2^2 \\ &\leq \frac{k^2\gamma^2}{rs^2} + \frac{4k^3(1-\delta)s^2\gamma^2}{r^2s^2} \\ &= \frac{5\gamma^2 k}{(1-\delta)s^2}.\end{aligned}$$

□

Therefore, to bound $\mathrm{err}_1(\mathbf{A})$ or $\mathbf{u}_1$, it suffices to bound $\|\mathbf{A} - \mathbb{E}\mathbf{A}\|_2$. Moreover, the bounds we get by either method are within a constant factor of each other. By [37], if $s \gg \log^4(k)$, then $\mathbf{A}$ will concentrate well around $\mathbb{E}\mathbf{A}$. This bound was later improved by the following result from [55].

Lemma 11.17. *Suppose we have a random Erdos-Renyi graph $G(n,p)$ with adjacency matrix $\mathbf{B}$ where $np \geq \log(n)$. For any $\alpha \geq 1$ there exists a universal constant $C_1 = C_1(\alpha)$ such that with probability at least $1 - n^{-\alpha}$,*

$$\|\mathbf{B} - \mathbb{E}\mathbf{B}\|_2 \leq C_1\sqrt{np}.$$

More generally, assume that $\mathbf{B}$ is a $n \times n$ adjacency matrix where $\mathbf{B}_{i,j}$ is Bernoulli with probability $p_{i,j}$. This is sometimes referred to as the inhomogeneous Erdos-Renyi model $G(n,(p_{i,j})$. Let $p = \max_{i,j} p_{i,j}$. As discussed in [53], Lemma 11.17 extends to this

setting using this definition of p (see section 1.1). While this result applies directly to $n \times n$ adjacency matrices, we can easily extend this to $\mathbf{A}$. This will first require a basic lemma about the spectral norm of a structured block matrix.

Lemma 11.18. *Let* $\mathbf{D}$ *be a* $n_1 \times n_2$ *matrix. Suppose that* $\mathbf{C}$ *is a* $n \times n$ *block matrix of the form*

$$\mathbf{C} = \begin{pmatrix} 0 & \mathbf{D} \\ \mathbf{D}^T & 0 \end{pmatrix}.$$

Then $\|\mathbf{C}\|_2 = \|\mathbf{D}\|_2$.

Proof. Standard properties of singular values imply that $\|\mathbf{D^T D}\|_2 = \|\mathbf{D D^T}\|_2 = \|\mathbf{D}\|_2^2$. Moreover,

$$\mathbf{C}\mathbf{C}^T = \begin{pmatrix} \mathbf{D}\mathbf{D}^T & 0 \\ 0 & \mathbf{D}^T\mathbf{D} \end{pmatrix}.$$

Since the eigenvalues of a block diagonal matrix are given by the eigenvalues of all the blocks,

$$\|\mathbf{C}\|_2^2 = \|\mathbf{C}\mathbf{C}^T\|_2 = \max\{\|\mathbf{D^T D}\|_2, \|\mathbf{D D^T}\|_2\} = \|\mathbf{D}\|_2^2.$$

□

Theorem 11.19. *Let* $\mathbf{A}$ *be a* $k \times r$ *matrix where* $k \geq r$ *and* $\mathbf{A}_{i,j}$ *is Bernoulli with probability* s/k. *Then for all* $\alpha \geq 1$, *there exists a universal constant* $C_2 = C_2(\alpha)$ *such that with probability at least* $1 - (k+r)^{-\alpha}$,

$$\|\mathbf{A} - \mathbb{E}\mathbf{A}\|_2 \leq C_2\sqrt{s}.$$

Proof. $\mathbf{A}$ encodes the structure of a bipartite graph with $k+r$ vertices. After relabeling, we can denote these vertices as $v_1, \ldots, v_k, v_{k+1}, \ldots, v_{k+r}$ where the bipartite blocks are

given by $\{v_1, \ldots, v_k\}, \{v_{k+1}, \ldots, v_{k+r}\}$. The adjacency matrix $\mathbf{B}$ is therefore of the form

$$\mathbf{B} = \begin{pmatrix} 0 & \mathbf{A} \\ \mathbf{A}^T & 0 \end{pmatrix}.$$

Note that $\mathbf{B}$ comes from an inhomogeneous Erdos-Renyi graph $G(k+r, (p_{i,j}))$ where $p_{i,j}$ is zero if i and j are both in $\{1, \ldots, k\}$ or $\{k+1, \ldots, k+r\}$, and s/k otherwise. Therefore, $p = \max_{i,j} p_{i,j} = s/k$. By Lemma 11.17 (and the discussion following it), for all $\alpha > 0$ there exists some universal constant $C_1 = C_1(\alpha)$ such that with probability at least $1 - (k+r)^{-\alpha}$,

$$\|\mathbf{B} - \mathbb{E}\mathbf{B}\|_2 \leq C_1 \sqrt{\frac{(k+r)s}{k}} \leq \sqrt{2} C_1 \sqrt{s}.$$

Here, we used the fact that $r \leq k$. Note that $\mathbb{E}\mathbf{B}$ satisfies

$$\mathbb{E}\mathbf{B} = \begin{pmatrix} 0 & \mathbb{E}\mathbf{A} \\ \mathbb{E}\mathbf{A}^T & 0 \end{pmatrix}.$$

Therefore,

$$\begin{aligned} \|\mathbf{B} - \mathbb{E}\mathbf{B}\|_2 &= \left\| \begin{pmatrix} 0 & \mathbf{A} - \mathbb{E}\mathbf{A} \\ (\mathbf{A} - \mathbb{E}\mathbf{A})^T & 0 \end{pmatrix} \right\|_2 \\ &= \|\mathbf{A} - \mathbb{E}\mathbf{A}\|_2. \end{aligned}$$

This last equality holds by Lemma 11.18. Taking $C_2 = \sqrt{2} C_1$ we conclude the proof. □

Combining this with Lemma 11.15, we get the following theorem.

Theorem 11.20. *Suppose that $s \geq \log(n)$. Then for any $\alpha \geq 1$, there is a universal constant $C_2 = C_2(\alpha)$ such that for $\rho = \frac{k}{rs}$, with probability at least $1 - (k+r)^{-\alpha}$,*

$$\mathrm{err}_1(\mathbf{A}) \leq \frac{C_2^2 k}{(1-\delta)s}.$$

Remark 11.21. *Empirically, the same bound holds for other methods of generating* $\mathbf{G}$*. If we choose the non-zero support of each column by selecting* s *indices with or without replacement from* $\{1, \ldots, k\}$*, then we conjecture that the same theorem holds. Unfortunately, standard concentration inequalities are not enough to prove this result in such settings.*

11.3.3 Regularized Bernoulli Gradient Codes

Bernoulli codes have two issues when $s < \log(k)$. First, each column only computes s tasks in expectation, but may have columns with degree up to $s + \log(k)$. If $s \geq \log(k)$, this is not an issue as this gives us sparsity that is $O(s)$. When $s < \log(k)$, however, this may be an issue. Second, if $s \ll \log(k)$, $\mathbf{A}$ may not concentrate around $\mathbb{E}\mathbf{A}$. For example, if $s = O(1)$ then by [51],

$$\|\mathbf{A}\|_2 = (1 + o(1))\sqrt{\frac{\log k}{\log \log k}}.$$

On the other hand, $\mathbb{E}\mathbf{A} = p\mathbf{1}_{k \times r}$ so $\|\mathbb{E}\mathbf{A}\|_2 = p\sqrt{kr} = \sqrt{1-\delta}s$. For $s \ll \log(k)$, this implies that $\mathbf{A}$ does not concentrate as well. Therefore we cannot use Lemma 11.15 to bound $\text{err}(\mathbf{A})$.

Both of these issues have the same cause: vertices whose degree is too large. Fortunately, this issue of enforcing concentration of sparse graphs has been studied and partially resolved in [53]. They show that by appropriate regularization of graphs, we can improve their concentration in the sparse setting.

Theorem 11.22 ([53])**.** *Let* $\mathbf{B}$ *be a random graph from the inhomogeneous Erdos-Renyi model* $G(n, (p_{i,j}))$ *and let* $p = \max_{i,j} p_{i,j}$*. For any* $\alpha \geq 1$*, the following holds with probability at least* $1 - n^{-\alpha}$*. Take all vertices of* $\mathbf{B}$ *with degree larger than* $2np$ *and*

reduce the weights of the edges incident to those vertices in any way such that they have degree at most np. Let $\mathbf{B}'$ *denote the resulting graph. Then*

$$\|\mathbf{B}' - \mathbb{E}\mathbf{B}\|_2 \leq C_3 r^{3/2}\sqrt{np}.$$

Here C_3 is a universal constant. Note that this regularization can be performed analogously on $\mathbf{A}$. To form $\mathbf{A}'$, we simply look at all columns with degree more than $2s$ and change entries in those columns from 1 to 0 until these columns have degree s. This satisfies the criterion in the above theorem. We can then use an almost identical version of the proof of Theorem 11.19 to prove the following theorem.

Theorem 11.23. *There is a universal constant* $C_4 = C_4(\alpha)$ *such that for any* $\alpha \geq 1$, $s \geq 1$, *with probability at least* $1 - (k+r)^{-\alpha}$,

$$\|\mathbf{A}' - \mathbb{E}\mathbf{A}\|_2 \leq C_3 \alpha^{3/2}\sqrt{s}.$$

We can combine this with Lemma 11.15 to derive the following theorem concerning $\text{err}(\mathbf{A}')$. As before, this bound applies for both the one-step decoding and the optimal decoding.

Theorem 11.24. *For any* $\alpha \geq 1, s \geq 1$, *and letting* $\rho = \frac{k}{rs}$, *with probability at least* $1 - (k+r)^{-\alpha}$,

$$\text{err}_1(\mathbf{A}') \leq \frac{C_3^2 \alpha^3 k}{(1-\delta)s}.$$

This approach ensures that each compute node computes at most $2s$ tasks and that our error bound works for all s. Note that in practice, we cannot form $\mathbf{A}'$ from $\mathbf{A}$ as we do not know $\mathbf{A}$ a priori. Therefore we cannot tell the compute nodes to compute the functions corresponding to $\mathbf{A}'$. Instead, we can regularize $\mathbf{G}$ in the same manner to

obtain $\mathbf{G}'$ in the following way such that we can apply Theorem 11.24 above. We refer to this code as the *regularized Bernoulli Gradient Code*, (rBGC).

The construction is simple. We initialize $\mathbf{G}$ with each entry Bernoulli(s/k). For each column j with more than $2s$ non-zero entries, we randomly set entries to 0 until it has s non-zero entries. A detailed algorithm is provided below.

Algorithm 7: Regularized Bernoulli Gradient Code.

Input : n, k, s.
Output: A $k \times n$ regularized function assignment matrix $\mathbf{G}'$ with max degree $\leq 2s$

$\mathbf{G}' = \mathbf{0}_{k \times n}$.;
for $j = 1$ **to** n **do**
    $d = 0$;
    for $i = 1$ **to** k **do**
        $\mathbf{G}'_{i,j} = Bernoulli(s/k)$;
        $d = d + \mathbf{G}'_{i,j}$;
    end
    if $d > 2s$ **then**
        while $d > s$ **do**
            remove a random edge from column j;
            $d = d - 1$;
        end
    end
end
return $\mathbf{G}'$;

11.4 Simulations

11.4.1 Decoding Error of Various Coding Schemes

In this section we compare the empirical decoding error of the FRCs and the BGCs. Recall that we gave two decoding methods that achieve the optimal decoding error

$$\text{err}(\mathbf{A}) = \min_{\mathbf{x}} \|\mathbf{A}\mathbf{x} - \mathbf{1}_k\|_2^2$$

and the one-step decoding error

$$\text{err}_1(\mathbf{A}) = \left\| \frac{k}{rs}\mathbf{A}\mathbf{1}_r - \mathbf{1}_k \right\|_2^2.$$

We also compare FRCs and BGCs to a coding scheme proposed in [78]. There, Raviv et al. consider the scheme where $\mathbf{G}$ is an s-regular expander. They show that for all $k \times r$ submatrices $\mathbf{A}$,

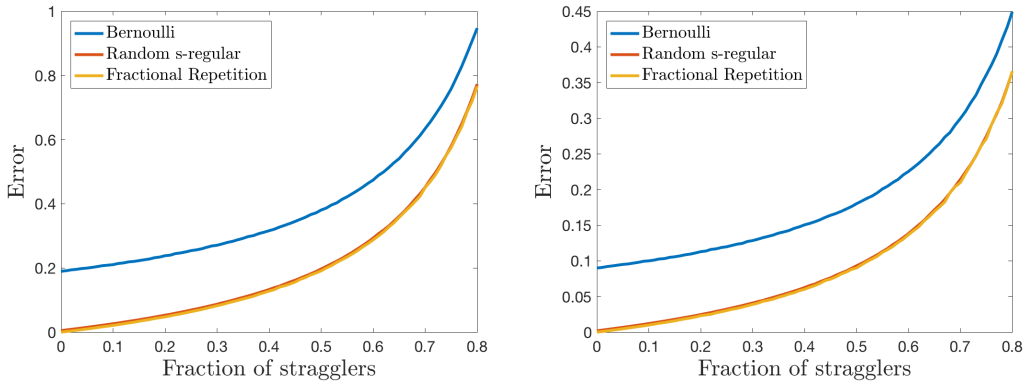
$$\text{err}_1(\mathbf{A}) \leq \frac{\lambda(\mathbf{G})^2}{s^2}\frac{\delta k}{(1-\delta)}.$$

Here, $\lambda(\mathbf{G}) = \max\{|\lambda_2|, |\lambda_k|\}$, where the eigenvalues of $\mathbf{G}$ are given by

$$\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_k.$$

We would like to construct $\mathbf{G}$ to have λ as small as possible. This is achieved by *Ramanujan graphs*. In practice, constructing expander graphs with small values of λ is difficult. By taking a random s-regular graph, however, we can obtain can expander graph with high probability [63]. As $k \to \infty$, λ tends to the optimal value. In order to generate empirical data, we consider the setting where $\mathbf{G}$ is the adjacency matrix of a random s-regular graph.

Below, we plot the one-step and optimal decoding error $\text{err}(\mathbf{A})$ and $\text{err}_1(\mathbf{A})$ for these three schemes when $k = 100$ and the fraction of stragglers δ varies. In order to normalize the error, we plot $\text{err}(\mathbf{A})/k$ and $\text{err}_1(\mathbf{A})/k$. We take $\rho = \frac{k}{rs}$ in the one-step decoding.

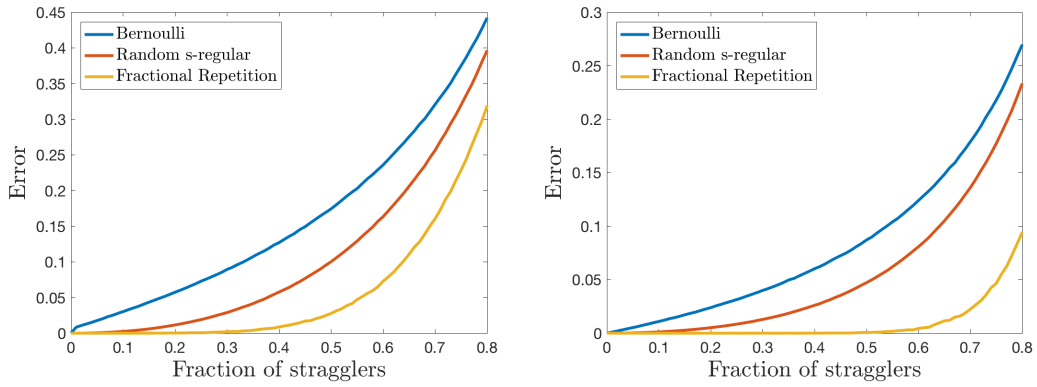


(a) $s = 5$ (b) $s = 10$

Figure 22: A plot of the average one-step error $\text{err}_1(\mathbf{A})/k$ over 5000 trials. We take $k = 100$, $r = (1 - \delta)k$ for varying δ. The figure on the left has $s = 5$ while the figure on the right has $s = 10$.

We see that under one-step decoding, FRCs and s-regular expanders perform extremely comparably. In this setting, BGCs seem to sacrifice some accuracy for simplicity. However, FRCs are also computationally simple and perform as well as taking s-regular expanders in the average case.

These plots show that if we instead consider optimal decoding, then FRCs greatly outperform the other two methods. In particular, FRCs can achieve zero optimal decoding error even with a non-trivial fraction of stragglers. If $s = 10$, then we can achieve close to zero error even with half of the compute nodes being stragglers.



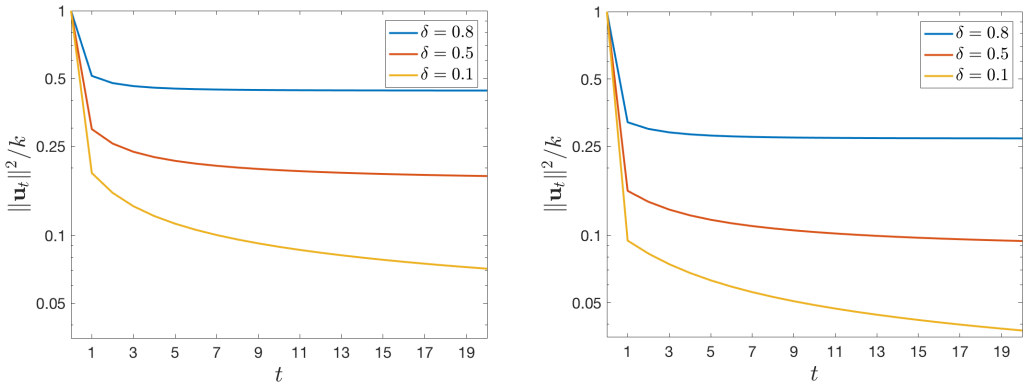
(a) $s = 5$ (b) $s = 10$

Figure 23: A plot of the average optimal decoding error $\text{err}(\mathbf{A})/k$ over 5000 trials. We take $k = 100$, $r = (1 - \delta)k$ for varying δ. The figure on the left has $s = 5$ while the figure on the right has $s = 10$.

11.4.2 Algorithmic Decoding Error of Bernoulli Coding

In this section we give empirical one-step error rates for varying sizes of r and sparsity s of the matrix $\mathbf{A}$ for various coding schemes. Recall that we defined the algorithmic decoding error of $\mathbf{A}$ by a sequence of vectors $\mathbf{u}_t$. Here, $\|\mathbf{u}_1\|_2^2$ corresponds to the one-step decoding error, while $\|\mathbf{u}_t\|_2^2$ converges to the optimal decoding error.

We consider the setting where $\mathbf{G}$ is constructed via a BGC. We fix $k = 100$ and take varying values of δ and s, letting $r = (1 - \delta)s$. We then calculate the average value of $\|\mathbf{u}_t\|_2^2/k$ based on a Monte Carlo simulation for increasing values of t. We set $\rho = \|\mathbf{A}\|_2^2$. The results are below.



(a) $s = 5$ (b) $s = 10$

Figure 24: The average value of $\|\mathbf{u}_t\|_2^2/k$ of a BGC for $\delta \in \{0.1, 0.2, 0.3, 0.5, 0.8\}$ and varying t for 5000 trials. The figure on the left plots the algorithmic error for sparsity $s = 5$, while the figure on the right plots the algorithmic error for sparsity $s = 10$.

Bibliography

[1] Ganesh Ananthanarayanan, Ali Ghodsi, Scott Shenker, and Ion Stoica. Effective straggler mitigation: Attack of the clones. In *NSDI*, volume 13, pages 185–198, 2013.

[2] Mihai Anitescu. Degenerate nonlinear programming with a quadratic growth condition. *SIAM Journal on Optimization*, 10(4):1116–1135, 2000.

[3] Yuichi Asahiro, Refael Hassin, and Kazuo Iwama. Complexity of finding dense subgraphs. *Discrete Applied Mathematics*, 121(1):15–26, 2002.

[4] Bernard A Asner, Jr. On the total nonnegativity of the Hurwitz matrix. *SIAM Journal on Applied Mathematics*, 18(2):407–414, 1970.

[5] Karl Johan Aström and Richard Murray. *Feedback systems: an introduction for scientists and engineers*. Princeton university press, 2010.

[6] Keith Ball. An elementary introduction to modern convex geometry. *Flavors of geometry*, 31:1–58, 1997.

[7] Stephen Becker, Emmanuel Candès, and Michael Grant. Templates for convex cone problems with applications to sparse signal recovery. *Mathematical programming computation*, 3(3):165–218, 2011.

[8] Walter Bergweiler and Alexandre Eremenko. Goldbergs constants. *Journal d'Analyse Mathématique*, 119(1):365–402, 2013.

[9] Vincent Blondel. *Simultaneous stabilization of linear systems*. 1994.

[10] Joseph Frédéric Bonnans and Alexander Ioffe. Second-order sufficiency and quadratic growth for nonisolated minima. *Mathematics of Operations Research*, 20(4):801–817, 1995.

[11] Nigel Boston. On the Belgian chocolate problem and output feedback stabilization: Efficacy of algebraic methods. In *Communication, Control, and Computing (Allerton), 2012 50th Annual Allerton Conference on*, pages 869–870. IEEE, 2012.

[12] Olivier Bousquet and André Elisseeff. Stability and generalization. *J. Mach. Learn. Res.*, 2:499–526, March 2002.

[13] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine Learning*, 3(1):1–122, 2011.

[14] Paul Bradley and Olvi Mangasarian. K-plane clustering. *Journal of Global Optimization*, 16(1):23–32, 2000.

[15] René Brandenberg. Radii of regular polytopes. *Discrete & Computational Geometry*, 33(1):43–55, 2005.

[16] Sébastien Bubeck et al. Convex optimization: Algorithms and complexity. *Foundations and Trends® in Machine Learning*, 8(3-4):231–357, 2015.

[17] James Burke, Didier Henrion, Adrian Lewis, and Michael Overton. Analysis of a Belgian chocolate stabilization problem. *LAAS-CNRS Research Report*, 5164, 2005.

[18] Elizabeth Burnside, Jesse Davis, Vítor Santos Costa, Inês de Castro Dutra, Charles Kahn Jr, Jason Fine, and David Page. Knowledge discovery from structured mammography reports using inductive logic programming. In *AMIA Annual Symposium Proceedings*, volume 2005, page 96. American Medical Informatics Association, 2005.

[19] Emmanuel Candès and Benjamin Recht. Exact matrix completion via convex optimization. *Communications of the ACM*, 55(6):111–119, 2012.

[20] YoungJung Chang and Nikolaos Sahinidis. Global optimization in stabilizing controller design. *Journal of Global Optimization*, 38(4):509–526, 2007.

[21] Jianmin Chen, Rajat Monga, Samy Bengio, and Rafal Jozefowicz. Revisiting distributed synchronous sgd. *arXiv preprint arXiv:1604.00981*, 2016.

[22] João Paulo Costeira and Takeo Kanade. A multibody factorization method for independently moving objects. *International Journal of Computer Vision*, 29(3):159–179, 1998.

[23] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, et al. Large scale distributed deep networks. In *Advances in Neural Information Processing Systems*, pages 1223–1231, 2012.

[24] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[25] Luc Devroye and Terry Wagner. Distribution-free performance bounds for potential function rules. *IEEE Transactions on Information Theory*, 25(5):601–604, 1979.

[26] David Donoho. For most large underdetermined systems of linear equations the minimal 1-norm solution is also the sparsest solution. *Communications on pure and applied mathematics*, 59(6):797–829, 2006.

[27] Sanghamitra Dutta, Viveck Cadambe, and Pulkit Grover. Short-dot: Computing large linear transforms distributedly using coded short dot products. In *Advances In Neural Information Processing Systems*, pages 2100–2108, 2016.

[28] Cynthia Dwork. Differential privacy. In *33rd International Colloquium on Automata, Languages and Programming, part II (ICALP 2006)*, volume 4052, pages 1–12, Venice, Italy, July 2006. Springer Verlag.

[29] Cynthia Dwork, Vitaly Feldman, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Aaron Leon Roth. Preserving statistical validity in adaptive data analysis. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing*, pages 117–126. ACM, 2015.

[30] Ehsan Elhamifar. High-rank matrix completion and clustering under self-expressive models. In *Advances in Neural Information Processing Systems*, pages 73–81, 2016.

[31] Ehsan Elhamifar and René Vidal. Sparse subspace clustering. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 2790–2797. IEEE, 2009.

[32] Andre Elisseeff, Theodoros Evgeniou, and Massimiliano Pontil. Stability of randomized learning algorithms. *J. Mach. Learn. Res.*, 6:55–79, December 2005.

[33] Charles Elkan and Keith Noto. Learning classifiers from only positive and unlabeled data. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 213–220. ACM, 2008.

[34] Brian Eriksson, Laura Balzano, and Robert Nowak. High-rank matrix completion. In *AISTATS*, pages 373–381, 2012.

[35] Brian Eriksson, Paul Barford, Joel Sommers, and Robert Nowak. DomainImpute: Inferring unseen components in the internet. In *INFOCOM, 2011 Proceedings IEEE*, pages 171–175. IEEE, 2011.

[36] William Fithian and Trevor Hastie. Finite-sample equivalence in statistical models for presence-only data. *The annals of applied statistics*, 7(4):1917, 2013.

[37] Zoltán Füredi and János Komlós. The eigenvalues of random symmetric matrices. *Combinatorica*, 1(3):233–241, 1981.

[38] Saeed Ghadimi and Guanghui Lan. Stochastic first-and zeroth-order methods for nonconvex stochastic programming. *SIAM Journal on Optimization*, 23(4):2341–2368, 2013.

[39] Alon Gonen and Shai Shalev-Shwartz. Fast rates for empirical risk minimization of strict saddle problems. *arXiv preprint, arXiv:1701.04271*, 2017.

[40] Amit Gruber and Yair Weiss. Multibody factorization with uncertainty and missing data using the em algorithm. In *Computer Vision and Pattern Recognition*, volume 1. IEEE, 2004.

[41] He Guannan, Wang Long, Xia Bican, and Yu Wensheng. Stabilization of the Belgian chocolate system via low-order controllers. In *Control Conference, 2007. CCC 2007. Chinese*, pages 88–92. IEEE, 2007.

[42] Moritz Hardt and Tengyu Ma. Identity matters in deep learning. *CoRR*, abs/1611.04231, 2016.

[43] Moritz Hardt, Ben Recht, and Yoram Singer. Train faster, generalize better: Stability of stochastic gradient descent. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, pages 1225–1234, 2016.

[44] Wei Hong, John Wright, Kun Huang, and Yi Ma. Multiscale hybrid linear models for lossy image representation. *IEEE Transactions on Image Processing*, 15(12):3655–3671, 2006.

[45] Alexander Ioffe. On sensitivity analysis of nonlinear programs in banach spaces: the approach via composite unconstrained optimization. *SIAM Journal on Optimization*, 4(1):1–43, 1994.

[46] Rie Johnson and Tong Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in Neural Information Processing Systems*, pages 315–323, 2013.

[47] Kenichi Kanatani. Motion segmentation by subspace separation and model selection. In *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*, volume 2, pages 586–591 vol.2, 2001.

[48] Hamed Karimi, Julie Nutini, and Mark Schmidt. *Linear Convergence of Gradient and Proximal-Gradient Methods Under the Polyak-Łojasiewicz Condition*, pages 795–811. Springer International Publishing, Cham, 2016.

[49] Kenji Kawaguchi. Deep learning without poor local minima. In *Advances in Neural Information Processing Systems*, pages 586–594, 2016.

[50] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint, arXiv:1609.04836*, 2016.

[51] Michael Krivelevich and Benny Sudakov. The largest eigenvalue of sparse random graphs. *Comb. Probab. Comput.*, 12(1):61–72, January 2003.

[52] Ilja Kuzborskij and Christoph Lampert. Data-dependent stability of stochastic gradient descent. *arXiv preprint, arXiv:1703.01678*, 2017.

[53] Can M Le, Elizaveta Levina, and Roman Vershynin. Concentration and regularization of random graphs. *Random Structures & Algorithms*, 2017.

[54] Kangwook Lee, Maximilian Lam, Ramtin Pedarsani, Dimitris Papailiopoulos, and Kannan Ramchandran. Speeding up distributed machine learning using codes. In *Information Theory (ISIT), 2016 IEEE International Symposium on*, pages 1143–1147. IEEE, 2016.

[55] Jing Lei, Alessandro Rinaldo, et al. Consistency of spectral clustering in stochastic block models. *The Annals of Statistics*, 43(1):215–237, 2015.

[56] Songze Li, Mohammad Ali Maddah-Ali, and A Salman Avestimehr. Coded mapreduce. In *Communication, Control, and Computing (Allerton), 2015 53rd Annual Allerton Conference on*, pages 964–971. IEEE, 2015.

[57] Henry Lin, Max Tegmark, and David Rolnick. Why does deep and cheap learning work so well? *Journal of Statistical Physics*, 168(6):1223–1247, 2017.

[58] Junhong Lin, Raffaello Camoriano, and Lorenzo Rosasco. Generalization properties and implicit regularization for multiple passes sgm. In *International Conference on Machine Learning*, 2016.

[59] Bing Liu, Yang Dai, Xiaoli Li, Wee Sun Lee, and Philip Yu. Building text classifiers using positive and unlabeled examples. In *Data Mining, 2003. ICDM 2003. Third IEEE International Conference on*, pages 179–186. IEEE, 2003.

[60] Tongliang Liu, Gábor Lugosi, Gergely Neu, and Dacheng Tao. Algorithmic stability and hypothesis complexity. *arXiv preprint, arXiv:1702.08712*, 2017.

[61] Stanisław Łojasiewicz. A topological property of real analytic subsets. *Coll. du CNRS, Les équations aux dérivées partielles*, 117:87–89, 1963.

[62] J Kenji López-Alt. *The Food Lab: Better Home Cooking Through Science*. WW Norton & Company, 2015.

[63] Alexander Lubotzky. Expander graphs in pure and applied mathematics. *Bulletin of the American Mathematical Society*, 49(1):113–162, 2012.

[64] Horia Mania, Xinghao Pan, Dimitris Papailiopoulos, Benjamin Recht, Kannan Ramchandran, and Michael I Jordan. Perturbed iterate analysis for asynchronous stochastic optimization. *SIAM Journal on Optimization*, 27(4):2202–2229, 2017.

[65] Pertti Mattila. *Geometry of sets and measures in Euclidean spaces: fractals and rectifiability*, volume 44. Cambridge university press, 1999.

[66] Sayan Mukherjee, Partha Niyogi, Tomaso Poggio, and Ryan Rifkin. Learning theory: stability is sufficient for generalization and necessary and sufficient for consistency of empirical risk minimization. *Advances in Computational Mathematics*, 25(1):161–193, 2006.

[67] Arvind Neelakantan, Luke Vilnis, Quoc V Le, Ilya Sutskever, Lukasz Kaiser, Karol Kurach, and James Martens. Adding gradient noise improves learning for very deep networks. *arXiv preprint arXiv:1511.06807*, 2015.

[68] Yu Nesterov. Efficiency of coordinate descent methods on huge-scale optimization problems. *SIAM Journal on Optimization*, 22(2):341–362, 2012.

[69] Yurii Nesterov. *Introductory lectures on convex optimization: A basic course*, volume 87. Springer Science & Business Media, 2013.

[70] Andrew Ng, Michael Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. In *Advances in neural information processing systems*, pages 849–856, 2002.

[71] Kobbi Nissim and Uri Stemmer. On the generalization properties of differential privacy. *CoRR, abs/1504.05800*, 2015.

[72] Vijay Patel, Girish Deodhare, and T Viswanath. Some applications of randomized algorithms for control system design. *Automatica*, 38(12):2085–2092, 2002.

[73] Jennie Pearce and Mark Boyce. Modelling distribution and abundance with presence-only data. *Journal of applied ecology*, 43(3):405–412, 2006.

[74] Daniel Pimentel-Alarcón, Laura Balzano, Roummel Marcia, R Nowak, and Rebecca Willett. Group-sparse subspace clustering with missing data. In *Statistical Signal Processing Workshop (SSP), 2016 IEEE*, pages 1–5. IEEE, 2016.

[75] Daniel Pimentel-Alarcon and Robert Nowak. The information-theoretic requirements of subspace clustering with missing data. In *International Conference on Machine Learning*, pages 802–810, 2016.

[76] Hang Qi, Evan Sparks, and Ameet Talwalkar. Paleo: A performance model for deep neural networks. In *International Conference on Learning Representations*, 2017.

[77] Shankar Rao, Roberto Tron, René Vidal, and Yi Ma. Motion segmentation via robust subspace separation in the presence of outlying, incomplete, or corrupted trajectories. In *Computer Vision and Pattern Recognition*, pages 1–8. IEEE, 2008.

[78] Netanel Raviv, Itzhak Tamo, Rashish Tandon, and Alexandros Dimakis. Gradient coding from cyclic mds codes and expander graphs. *arXiv preprint arXiv:1707.03858*, 2017.

[79] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. Hogwild: A

lock-free approach to parallelizing stochastic gradient descent. In *Advances in neural information processing systems*, pages 693–701, 2011.

[80] Amirhossein Reisizadeh, Saurav Prakash, Ramtin Pedarsani, and Salman Avestimehr. Coded computation over heterogeneous clusters. In *Information Theory (ISIT), 2017 IEEE International Symposium on*, pages 2408–2412. IEEE, 2017.

[81] David Rumelhart, Geoffrey Hinton, Ronald Williams, et al. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

[82] Hong Ryoo and Nikolaos Sahinidis. A branch-and-reduce approach to global optimization. *Journal of Global Optimization*, 8(2):107–138, 1996.

[83] Nihar Shah, Kangwook Lee, and Kannan Ramchandran. When do redundant requests reduce latency? *IEEE Transactions on Communications*, 64(2):715–722, 2016.

[84] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

[85] Shai Shalev-Shwartz, Ohad Shamir, Nathan Srebro, and Karthik Sridharan. Learnability, stability and uniform convergence. *Journal of Machine Learning Research*, 11(Oct):2635–2670, 2010.

[86] Mahdi Soltanolkotabi and Emmanuel Cands. A geometric analysis of subspace clustering with outliers. *Ann. Statist.*, 40(4):2195–2238, 08 2012.

[87] Mahdi Soltanolkotabi, Ehsan Elhamifar, and Emmanuel Candés. Robust subspace clustering. *The Annals of Statistics*, 42(2):669–699, 2014.

[88] Michel Talagrand. Concentration of measure and isoperimetric inequalities in product spaces. *Publications Mathematiques de l'IHES*, 81(1):73–205, 1995.

[89] Rashish Tandon, Qi Lei, Alexandros Dimakis, and Nikos Karampatziakis. Gradient coding: Avoiding stragglers in distributed learning. In *International Conference on Machine Learning*, pages 3368–3376, 2017.

[90] Jan Verschelde. Algorithm 795: Phcpack: A general-purpose solver for polynomial systems by homotopy continuation. *ACM Transactions on Mathematical Software (TOMS)*, 25(2):251–276, 1999.

[91] René Vidal and Richard Hartley. Motion segmentation with missing data using PowerFactorization and GPCA. In *Computer Vision and Pattern Recognition*, volume 2. IEEE, 2004.

[92] Rene Vidal, Yi Ma, and Shankar Sastry. Generalized principal component analysis GPCA. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(12):1945–1959, 2005.

[93] Ulrike Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416, 2007.

[94] Da Wang, Gauri Joshi, and Gregory Wornell. Efficient task replication for fast response times in parallel computation. In *ACM SIGMETRICS Performance Evaluation Review*, volume 42, pages 599–600. ACM, 2014.

[95] Yining Wang, Yu-Xiang Wang, and Aarti Singh. Graph connectivity in noisy

sparse subspace clustering. In *Artificial Intelligence and Statistics*, pages 538–546, 2016.

[96] Yu-Xiang Wang and Huan Xu. Noisy sparse subspace clustering. *J. Mach. Learn. Res.*, 17(1):320–360, January 2016.

[97] Gill Ward, Trevor Hastie, Simon Barry, Jane Elith, and John Leathwick. Presence-only data and the em algorithm. *Biometrics*, 65(2):554–563, 2009.

[98] Stephen Wright. *Primal-dual interior-point methods*. SIAM, 1997.

[99] Bo Xie, Yingyu Liang, and Le Song. Diverse neural network learns true target functions. In *Artificial Intelligence and Statistics*, pages 1216–1224, 2017.

[100] Neeraja Yadwadkar and Wontae Choi. Proactive straggler avoidance using machine learning. *White paper, University of Berkeley*, 2012.

[101] Allen Yang, John Wright, Yi Ma, and Shankar Sastry. Unsupervised segmentation of natural images via lossy data compression. *Computer Vision and Image Understanding*, 110(2):212–225, 2008.

[102] Congyuan Yang, Daniel Robinson, and René Vidal. Sparse subspace clustering with missing entries. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning*, volume 37, pages 2463–2472, 2015.

[103] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. *HotCloud*, 10(10-10):95, 2010.

[104] Amy Zhang, Nadia Fawaz, Stratis Ioannidis, and Andrea Montanari. Guess who rated this movie: Identifying users through subspace clustering. In *Proceedings of the Twenty-Eighth Conference on Uncertainty in Artificial Intelligence*, pages 944–953, 2012.

[105] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *arXiv preprint, arXiv:1611.03530*, 2016.

[106] Pan Zhou and Jiashi Feng. The landscape of deep learning algorithms. *arXiv preprint, arXiv:1705.07038*, 2017.

[107] Anastasios Zouzias and Nikolaos M Freris. Randomized extended kaczmarz for solving least squares. *SIAM Journal on Matrix Analysis and Applications*, 34(2):773–793, 2013.