The Wonderful World of Constraints in Learning and Vision

by

Sathya Narayanan Ravi

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

Doctor of Philosophy

(Computer Sciences)

at the

UNIVERSITY OF WISCONSIN–MADISON

2019

Date of final oral examination: 08/08/2019

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

Vikas Singh (Advisor), Professor, Biostatistics and Medical Informatics
Xiaojin Zhu, Professor, Computer Sciences
Karl Rohe, Associate Professor, Statistics
Mohit Gupta, Assistant Professor, Computer Sciences

Acknowledgments

I will forever be indebted to the time and effort that Vikas has spent to make this dissertation possible. I thank Jerry, Karl, and Mohit for agreeing to listen to what I had to say, twice! A big thanks to the math department for believing in me as a TA during my early graduate school days when I had no idea what I wanted to work on, specifically. I am grateful to the funding agencies and grants that helped my research (and to live) during my graduate studies at UW such as UW CPCP (U54 AI117924), University of Wisconsin ADRC (P50 AG033514), and American Family Insurance, NIH R01 AG040396, NSF CCF 1320755, and NSF CAREER RI 1252725. Thanks to friends, and family for their (almost surely) continuous support. I dedicate this thesis to Sadhguru.

Contents

Contents ii

List of Tables v

List of Figures vi

Abstract viii

1 Introduction 1
- 1.1 Solving a ML Model . . . 5
- 1.2 Scope of this thesis . . . 7
- 1.3 Outline . . . 11
- 1.4 Notations . . . 12

2 Mathematical Background 13
- 2.1 The landscape of Numerical Optimization . . . 13

3 Making Filter Flow Practical 32
- 3.1 Introduction . . . 32
- 3.2 A brief overview of Filter Flow . . . 36
- 3.3 Algorithmic Reformulations . . . 37
- 3.4 Analysis of Algorithm and Convergence . . . 42
- 3.5 C_f Calculations . . . 44
- 3.6 Case Studies . . . 46
- 3.7 Experiments . . . 48
- 3.8 Conclusions . . . 53

4 Coresets for Nonsmooth Problems 54

4.1 Introduction . . . 54
4.2 Preliminary Concepts . . . 60
4.3 Convergence Results . . . 62
4.4 Approximable Nonsmooth Problems . . . 68
4.5 Experiments . . . 84
4.6 Discussion . . . 90

5 Explicitly Imposing Constraints in Deep Networks 92
5.1 Introduction . . . 92
5.2 Do we need to impose constraints? . . . 94
5.3 First Order Methods: Two Representatives . . . 96
5.4 Categorizing "Generic" Constraints for CG . . . 98
5.5 Path Norm Constraints in Deep Learning . . . 103
5.6 Experiments . . . 106
5.7 Conclusions . . . 113

6 Experimental Design for Sparse Models 114
6.1 Introduction . . . 114
6.2 Preliminaries . . . 116
6.3 Our proposed formulations . . . 118
6.4 Experiments . . . 128
6.5 Conclusions . . . 134

7 Robust Blind Deconvolution via Mirror Descent 135
7.1 Introduction . . . 135
7.2 The Blind Deconvolution problem . . . 138
7.3 The PRIDA Algorithm . . . 140
7.4 Convergence, Robustness, and More . . . 144
7.5 Experiments . . . 149
7.6 Conclusion . . . 153

8 Conclusions 154
8.1 Learning Locality Sensitive Filter Flow Maps . . . 155
8.2 Coreset based Sampling and Applications . . . 155
8.3 Explicitly Imposing Probabilistic Constraints . . . 156
8.4 Faster Finetuning via Experimental Design . . . 157

8.5 Robustly Learning Kernels for Blind Deconvolution 158

References **159**

List of Tables

4.1 A summary of the relevant quantities derived for each problem described in the previous sections. 87

7.1 Average endpoint error across the dataset in Levin et al. (2009) for varying levels of noise. 151

List of Figures

1.1 Removing Perspective distortion in images by identifying planar regions 3
1.2 Scope of this thesis from an optimization perspective 9

2.1 Illustration of one iteration of PGD Algorithm 3 and CG Algorithm 4. . . 27

3.1 Optical flow results on the MPI Sintel dataset. 50
3.2 Optical flow results on the Middlebury dataset 50
3.3 Optical flow results on the test set of MPI Sintel dataset. 51
3.4 Stereo results: (From left to right) First row plots show the evaluation on Middlebury 2011 dataset 52

4.1 Convergence results for Alg. 6 (red) and randomized smoothing of Lan (2013) (blue) on the SVM datasets. 85
4.2 Convergence results for Alg. 6 using bisection line search. 86
4.3 Results on the size of coreset for the multiway graph cuts problems . . . 88

5.1 Illustration of the linear optimization Oracle for Path-CG algorithm . . . 105
5.2 Performance of CG on ResNet-32 on CIFAR10 dataset 108
5.3 Performance of Path-CG vs SGD on a 2-layer fully connected network on four datasets 109
5.4 Illustrates the task of image inpainting overall pipeline. 111
5.5 Qualitative results to compare the performance of CG and SGD for image inpainting task 112
5.6 Quantitative results for image inpainting using MSE/SSIM/FID 112

6.1 Variable selection of Lasso estimate β_1^* 119
6.2 Errors in consistent selection of correct features (derived from full model) for *prostate* and *lars* datasets 130

a Prostate dataset: error in feature selection vs Budgets 130
b Prostate dataset: error in feature selection vs #Features 130
c Lars dataset: error in feature selection vs Budgets 130
d Lars dataset: error in feature selection vs #Features 130
6.3 Comparison of models from ED-S and ED-I algorithms using Zero and Sign inconsistency rate 131
a Zero inconsistency of ED-I 131
b Sign inconsistency of ED-I 131
c Zero inconsistency of ED-S 131
d Sign inconsistency of ED-S 131
6.4 Change in BIC and R^2 vs. full model; Dependent variable moments from reduced vs. full models 132
a Change in BIC score 132
b Change in R-squared score 132
c Moment of dependent variable – CDR 132
d Moment of dependent variable – ADAS-Cog change 132
6.5 Quantifying the difference between reduced and full models using p-values obtained after hypothesis testing. 133
a ED-I vs. full (high λ) 133
b ED-I vs. full (low λ) 133
c ED-S vs. full (high λ) 133
d ED-S vs. full (low λ) 133

7.1 Visualization of Projected Gradient Descent (PGD) and Mirror Descent (MD) on the probability simplex 142
7.2 Convergence rate comparison of PRIDA (in red) and Perrone et al. in Perrone and Favaro (2014) (in blue). 149
7.3 Qualitative deblurring results of PRIDA algorithm on noisy input in comparison with two standard baselines 150
7.4 Color image recovery in the presence of intensity noise 151

Abstract

Constraints in the context of Machine Learning or Computer Vision play a central role to obtain accurate models of the overall system or to simply prescribe its behavior under various unknown circumstances. Often, these constraints are solely attributed to the application that the model eventually serves. For example, image transformation tasks can be specified effectively using constraints arising from the physics of the scene being captured. In the machine learning context, constraining the amount of samples that a training algorithm has access to various theoretical and practical benefits, especially when the dataset size becomes large. In this setting, it is sometimes possible that the training algorithm can satisfy resource constraints, that is, the algorithm does not need to access all the samples in the dataset to be reasonably accurate. In other cases, constraints are used to ensure generalization in a statistical sense. For example, in the context of training deep networks, by constraining functions that measure sample complexity of the task, it is possible to train much simpler models without loss of accuracy.

In this thesis we provide algorithms that can handle a variety of constraints efficiently. First, we provide a parallel optimization framework based on convex analysis. We then use the framework to develop algorithms with provable convergence guarantees for various Computer Vision tasks. Second, we develop a novel algorithm to construct coresets of tasks with nonsmooth loss function with optimal convergence guarantee. Third, we provide a catalog of constraints that can be efficiently imposed while training deep neural networks. In the last two chapters, we develop algorithms that can incorporate constraints arising due to the geometry of the feasible set or loss function. Fourth, we show how to preserve sparsity under budget constraints in longitudinal studies in clinical trials. Finally, we show how the geometry of the constraint set in standard formulations of the blind deconvolution problem can be solved using a first order algorithm that is also noise robust.

Chapter 1

Introduction

Artificial Intelligence (AI), as a field of research emerged in the late 1950's led by the ELIZA project to understand/simulate conversations between a human and a computer (Weizenbaum et al., 1966). To accomplish this, the ELIZA program was designed to detect keywords in the text corresponding to the user input. Using these keywords, the human dialogue was decomposed according to a programmed set of logical rules. These logical rules were carefully designed in order to capture the richness of the vocabulary in "natural" sentences that may be encountered in a conversation: both in the input sentence from the user, as well as the human that ELIZA was intended to impersonate. Different human personalities could be simulated with different set of template keywords, and sentences, making ELIZA an important historical landmark in AI research.

This simple approach used in ELIZA was quite effective to show that computers can hold human-like conversations for a moderate duration of time, and in some ways still serves to define the frameworks upon which modern conversational agents are developed (Shah et al., 2016). The simplicity of the "pattern-matching" approach in ELIZA induced rapid progress towards tackling various problems such as image analysis, search, planning, and scheduling. In fact, the idea that computers can be useful to describe visual information was first introduced in this period as a summer project (Hartley and Zisserman, 2003). These led to a belief that success of/from AI was just around the corner.

Early AI systems tried to emulate the success of the ELIZA program and were built using symbolic processing units that could compute Boolean logical rules. Symbolic AI is based on predicate logic in which our knowledge of the world is described and

represented explicitly using symbols, with clearly defined relationships between symbols and entities in the real world (Shapiro, 1971). In symbolic AI, we have to provide an explicit description of human readable information that is relevant to the task. While this approach was well grounded in the theory of (first order) predicate logic (Hamilton, 1988), research showed that it was difficult to quantify uncertainty using only symbolic computation. Hence, slowly researchers working in this area began to appreciate and acknowledge the need for non-symbolic or statistical techniques. For instance, "interpolating" reasoning using the symbolic framework required large amount of computational resource that were not then available. On the other hand, statistical tools including stochastic processes, spline functions, regression analysis, Bayesian methods etc. provided an arguably simpler solution for interpolation tasks since they were data-dependent and hence started being incorporated in modeling AI systems. As the research problems became complex, there was an increasing need for a paradigm to automatically improve performance using relevant data without constant supervision, and this general line of inquiry which was related but also distinct from statistics was later termed as *Machine Learning* (ML).

In 1967, what may be thought of as the first ML algorithm called Nearest Neighbor (NN) was developed. NN had the ability to recognize patterns: given a set of objects as an input, when the NN algorithm is presented with a new object, it outputs the most similar object in its database. Subsequently, a refined version of the similarity search problem, called classification was introduced where the goal was to assign a label to (individual) data samples based on the labeled input data. Support Vector Machines, and more generally, kernel methods, were invented as a means to classify new data based on observed data, and was widely adopted for fraud detection (Vapnik, 1963; Hofmann et al., 2008; Wahba, 1990). ML was then defined as an *algorithmic* (or a systematic) methodology to make future predictions using past *data* (or experiences). Currently, the proliferation of core ideas from ML, and more generally, data analysis methods can be seen in various disciplines spanning biochemistry (Panteleev et al., 2018), medicine (Obermeyer and Emanuel, 2016), physics (Baldi et al., 2014), astronomy (Ball and Brunner, 2010), finance (Heaton et al., 2017), and mathematics (Raayoni et al., 2019).

At a high level, a ML approach involves first collecting data that may be useful to gain knowledge about a specific *task* of interest. The task may correspond to classification (Alpaydin, 2009), regression (Hastie et al., 2001), clustering (Jain and

Figure 1.1: The inclination of the trees in the right image vanishes after removing the perspective distortion introduced by the camera. Perspective distortion is due to the fact that shapes in the 3D scene get distorted when mapped to 2D plane in the camera acquired image. In this case, it is possible to "undo" the distortion simply by first identifying four points on the 2D image that lie on a plane in the 3D scene, and then solving a system of linear equations with just 8 unknown coefficients (Adams, 1972).

Dubes, 1988), and so on. We then attempt to accumulate information from this (large) dataset that is relevant to make (approximately) accurate predictions for the *unobserved* data – this step is often referred to as *learning*. While this viewpoint captures the essence of ML method, formalizing each of the components (viz., algorithm, data) can sometimes be nontrivial, and constitutes a bulk of modern day research in ML.

One particular domain where ML has proven to be tremendously effective is *Computer Vision (CV)*. In the early stages of research in CV, problems primarily focused on mathematically understanding the geometric relations between the content captured in images. For example, it is possible to remove the *perspective distortion* and improve the image quality by extracting planar regions from images, see Figure 1.1. As time progressed, this direction became bigger in scope encompassing a diverse set of core image analysis problems such as face recognition, foreground background segmentation, content-based image retrieval and so on. The first successful practical application of CV was in Optical Character Recognition (OCR) which was implemented by the United States Postal Service in 1982 (Ulvila, 1987).

Vision problems such as image registration (Shen and Davatzikos, 2001), optical flow (Horn and Schunck, 1981), stereo matching (Kanade and Okutomi, 1991) involve computing a mapping between pixels on multiple images of the same 3D scene. ML based modeling frameworks such as Markov Random Fields (Li, 1994), Gaussian

Mixture Models (Raja et al., 1998) are widely used in these settings since they allow us to model and capture semantic relationships between images, especially in the presence of background noise, occlusion, and deformations. Observe that for images with N (number of) pixels, computing such mappings naively would require up to N^2 operations in the worst case. But the worst case may not be very likely, for example, say our goal is to compute a mapping between left and right images in Figure 1.1. In this case, it is clear that the optimal mapping would be nearly identity. This observation was used for pre-processing in order to avoid computational redundancy and involves two key steps: (i) feature detection is first performed using image properties such as gradients (Dalal and Triggs, 2005) and such image properties are then (ii) *represented* in an appropriate feature space. The corresponding "representations" in the feature space, e.g., the widely used SIFT descriptors (Lowe, 2004), can serve a number of downstream goals depending on the task at hand (Duan et al., 2012; Andreopoulos and Tsotsos, 2013). Once features are derived from individual images, then an appropriate mathematical model is solved using ML algorithms. This two-phase approach remained the standard pipeline in vision for a long time, and in fact, still drives a number of applications today. There are two main drawbacks with the two phase approach, especially when the dataset size becomes large (Deng et al., 2009): (i) each task requires a specialized pre-processing or feature representation technique; (ii) learned models are difficult to maintain/update, that is, as we obtain new data, we may have to repeat or fine tune both the phases which could be computationally infeasible.

To remedy the drawbacks we may recall a classical result from statistics called the central limit theorem: informally, when the dataset size increases, the observed data provides a good approximation to the true (unknown) distribution. This implies that if we have collected enough training samples, then using a data agnostic pre-processing step may result in loss of information, that may be useful for the task at hand. In other words, the statistical viewpoint suggests that instead of using a *separate pre-processing step*, we should also learn the feature descriptors in tandem with learning the parameters of model associated with the task (when we have access to sufficiently large amounts of data for the task). Research in the last decade has shown that Artificial Neural Networks (ANNs) or more generally, Deep Neural Networks (DNNs) can be used to build end-to-end vision systems with little or no pre-processing required, especially for image data (Krizhevsky et al., 2012). The success of DNNs

has been partly attributed (Le et al., 2012; Li et al., 2016a) to availability of large amounts of data and faster computational resources like GPU, parallel processing power, and various hardware advancements in the last 20 years or so. In essence, by DNNs we refer to over-parameterized ML models that can be used to learn (possibly) a hierarchy of representations from data, which can be used for various data analysis tasks including object classification, detection, regression, and so on. At the same time, in order to fully realize the potential promise of ML for specific application domains, it is becoming increasingly clear that the technical development *cannot* be undertaken agnostic of domain knowledge. This will be our main motivation in this thesis, and we will explore this in more detail from an algorithmic perspective.

1.1 Solving a ML Model

ML models are most commonly described as a search for optimal parameters with respect to an *objective function* over an appropriately chosen *feasible* set. The feasible set represents the set of allowable parameters, and may also be used to mathematically encode various domain related information for the task at hand such as prior knowledge, constraints, or general purpose thumb-rules. In the context of ML, the objective function is referred as the *loss* function which indicates the effectiveness of the captured information (Rosasco et al., 2004). Any such description including the objective function and feasible set is formally referred to as an *optimization problem*, and a procedure to (approximately) solve the problem is called a numerical optimization algorithm. The earliest known optimization algorithm dates back to 1711 called Newton's method which was used to find the roots of a polynomial. In the early 19th century, optimization was mostly studied in the field of Operations Research (OR) in order to improve the efficiency of various military operations such as scheduling of trains to minimize wait time, transportation of goods via the shortest path and so on (Ferguson and Dantzig, 1956; Midler and Wollmer, 1969). Following traditional nomenclature in OR, we will use "programs" and "optimization problems" interchangeably. Before we start focusing on specific applications studied in this thesis, we will first discuss some basic terminologies and notations that will be used through out the thesis.

1.1.1 Numerical Optimization

Numerical optimization corresponds to computing a point that either minimizes or maximizes a function called the objective function (which is usually fixed apriori), or simply the objective subject to feasibility constraints. Optimization problems are usually in a standard template as follows:

$$\min_{x \in C} f(x). \tag{1.1}$$

In (1.1), we seek to minimize the objective function denoted by $f(\cdot)$ with the input/argument x, such that the output of the algorithm satisfies a representation of the constraints or regularization described by C.

The process of identifying the objective function f, the decision variable(s) x, and the constraint or feasible set C is called *formulating* the real world problem that we intend to solve. In general, there can be more than one way to formulate a given problem, and multiple numerical schemes to solve a given formulation. Identifying the formulation for a problem that can be solved using efficient numerical schemes is critical since it determines the extent to which the solution procedure can be used in practice. For example, in large scale settings, algorithms that only require matrix vector products can exploit the parallel architecture available in GPUs to provide accurate solutions in orders of magnitude less (wall clock) time.

The efficiency of an algorithm is calculated based on the number of iterations required to solve the given optimization problem. We call a solution x^* to an optimization problem *optimal*, if

$$f(x^*) \leqslant f(x), \tag{1.2}$$

for all possible $x \in C$. In cases where we are able to output an optimal solution x^* (as in (1.2)), x^* is called as an *exact* solution, and the procedure to do so as an exact algorithm. In scenarios when exact solutions are either impossible or computationally intractable to find, we relax our goal to find an *approximate* solution. An (additive) approximate solution x^+ to (1.1) is defined as,

$$f(x^+) \leqslant f(x) + \epsilon,$$

where $\epsilon > 0$ is some constant that does not dependent on any problem parameters

such as f, and C.

There are various standard templates of optimization problems that have been studied extensively both in theory and practice. For example, when f is linear, and C can be described using linear inequalities, then (1.1) is called as a Linear Program (LP). ML models where the *hinge* function may be used as the loss function, they can be cast as a LP. LPs can be solved efficiently in practice even when the number of decision variables exceeds a few millions (Gearhart et al., 2013). However, when the decision variables are required to be integers, the optimization is theoretically hard to solve and may not be possible to solve in practical instances. Hence, it is often the case that we need to *reformulate* our "standard" formulation of a ML model in order to develop efficient algorithms. Reformulation or relaxation is the process of identifying an alternative formulation of the original formulation, possibly equivalent, for algorithmic purposes. It turns out that several classes of integer programs can be reformulated as Semi Definite Programs (SDPs) that can be solved in polynomial time (Vandenberghe et al.), and shown to provide a provable approximation to the integer program (Goemans and Williamson, 1995). We call an optimization problem a SDP if f is linear and C can be represented using a set of linear inequalities intersected with the positive semidefinite cone. In fact, these standard templates are often used for solving ML models using off-the-shelf software packages like CPLEX, Gurobi and CVX (Anand et al., 2017; Grant and Boyd, 2014). For example, for binary classification task, the most commonly used model is maximum margin Support Vector Machine (SVM) formulated as a (convex) quadratic programming problem which is simply a special case of SDP. In the vision context, estimating homography or a projective transformation can be achieved by minimizing the *Reprojection error* subject to point-to-point correspondence in two or more images specified as linear equality constraints.

1.2 Scope of this thesis

Broadly speaking, the work described in this is located at the intersection of optimization algorithms and artificial intelligence in high dimensional and large scale settings. At a finer level, we will discuss a wide range of applications, and analyze the theoretical properties of the resulting algorithms. The primary hypothesis of this thesis can be phrased as,

Hypothesis 1. Specialized algorithms that can satisfy application specific constraints and regularization arising out of domain knowledge or applications are often crucial and/or beneficial to the overall performance.

For example, faster, although approximate algorithms are preferred for object detection/classification but not while analyzing MRI scans from a cohort study. Figure 1.2 provides a better illustration of the work in this thesis where we develop numerical algorithms for problems in the shaded regions. In this thesis, we will study efficiency and algorithmic issues in solving optimization problems with explicit constraints that are imposed in ML and vision problems. In this section, we will briefly review five different problems studied in this thesis.

1.2.1 Making Filter Flow Practical

This project is inspired by the work of Seitz and Baker which introduced the so-called Filter Flow model (Seitz and Baker, 2009). Filter flow finds the transformation relating a pair of (or multiple) images by identifying a large set of local linear filters. Imposing additional constraints on certain structural properties of these filters enables Filter Flow to serve as a general “one stop" construction for a spectrum of problems in vision: from optical flow to defocus to stereo to affine alignment. The idea is beautiful yet the benefits are not borne out in practice because of significant computational challenges. This issue makes most (if not all) deployments for practical vision problems out of reach. The key thrust of our work is to identify mathematically (near) equivalent reformulations of this model that can eliminate this serious limitation. We demonstrate via a detailed optimization-focused development that Filter Flow can indeed be solved fairly efficiently for a wide range of instantiations. We derive efficient algorithms, perform extensive theoretical analysis focused on convergence and parallelization and show how results competitive with the state of the art for many applications can be achieved with negligible application specific adjustments or post-processing. The actual numerical scheme is easy to understand and, implement (30 lines in Matlab) — this development enables Filter Flow to be a viable general solver and testbed for numerous applications in the community, going forward.

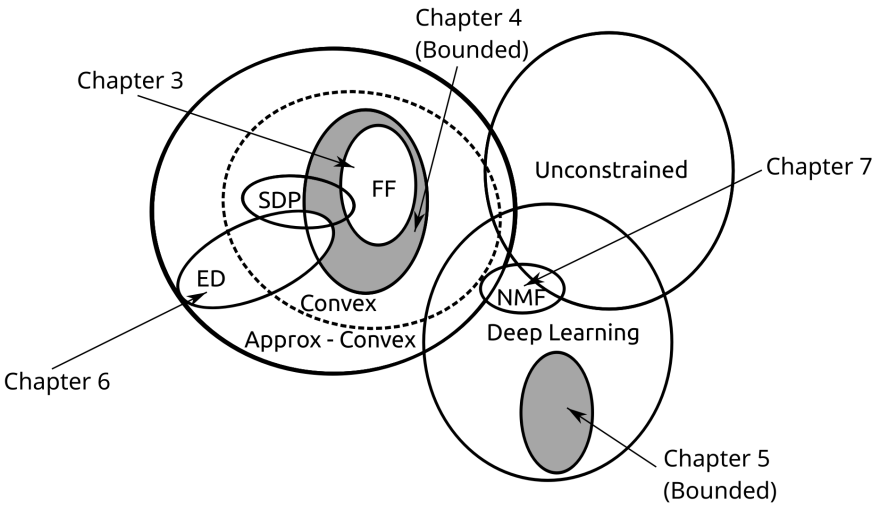


Figure 1.2: Illustrates the problems covered in the chapters 3-7 of this thesis – FF corresponds to the filter flow problems, discussed in Chapter 3; the shaded region corresponds to bounded convex/Deep Learning optimization problems, discussed in Chapter 4/5; ED corresponds to the Experimental Design for Sparse Models problem studied in Chapter 6; NMF corresponds to Nonnegative Matrix Factorization studied under the context of Blind Deconvolution in Chapter 7.

1.2.2 Coresets for Nonsmooth Problems

We present a new Conditional Gradient (CG) type algorithm that is applicable to minimization problems with a nonsmooth convex objective. We provide convergence bounds and show that the scheme yields so-called *coreset* results for various Machine Learning problems including 1-median, Balanced Development, Sparse PCA, Graph Cuts, and the ℓ_1-norm-regularized Support Vector Machine (SVM) among others. This means that the algorithm provides approximate solutions to these problems in time complexity bounds that are *not dependent* on the size of the input data. Our framework, motivated by a growing body of work on *sublinear* algorithms for various data analysis problems, is entirely deterministic and makes no use of smoothing or proximal operators. Apart from these theoretical results, we show experimentally that the algorithm is practical and in some cases also offers significant computational advantages on large problem instances. We provide an open source implementation that can be adapted for other problems that fit the overall structure.

1.2.3 Constrained Deep Learning

A number of results have recently demonstrated the benefits of incorporating various constraints when training deep architectures in vision and machine learning. The advantages range from guarantees for statistical generalization to better accuracy to compression. But support for general constraints within widely used libraries remains scarce and their broader deployment within many applications that can benefit from them remains under-explored. Part of the reason is that Stochastic gradient descent (SGD), the workhorse for training deep neural networks, *does not natively deal with constraints with global scope very well.* In this project, we use the CG algorithm, that has, thus far had limited applicability in training deep models. We show via rigorous analysis how various constraints can be naturally handled by modifications of this algorithm. We show a suite of *immediate benefits* that are possible — from training ResNets with fewer layers but better accuracy simply by substituting in our version of CG to faster training of GANs with 50% fewer epochs in image inpainting applications to provably better generalization guarantees using efficiently implementable forms of recently proposed regularizers.

1.2.4 Experimental Design for Sparse Models

Budget constrained optimal design of experiments is a well studied problem. Although the literature is very mature, not many strategies are available when these design problems appear in the context of sparse linear models commonly encountered in high dimensional machine learning. We study this budget constrained design where the underlying regression model involves a ℓ_1-regularized linear function. We propose two novel strategies: the first is motivated geometrically whereas the second is algebraic in nature. We obtain tractable algorithms for this problem which also hold for a more general class of sparse linear models. We perform a detailed set of experiments, on benchmarks and a large neuroimaging study, showing that the proposed models are effective in practice. The latter experiment suggests that these ideas may play a small role in informing enrollment strategies for similar scientific studies in the future.

1.2.5 Robust Blind Deconvolution via Mirror Descent

We revisit a classical problem in computer vision and image analysis called as Blind Deconvolution. In this problem, given a blurred image, the goal is to recover the associated sharp image and the blur kernel simultaneously. Although the literature in this area is mature, the algorithmic underpinnings of the problem is still not fully understood and is being actively researched. In this work, we propose an efficient algorithm, PRIDA for optimizing nonconvex loss functions that may be encountered in DL problems with simplex constraints. Theoretically, we analyze both the *convergence* and *robustness* properties of PRIDA. Empirically, we perform an extensive set of experiments to validate our theoretical findings on two standard datasets. Our results show that PRIDA beats the state-of-the-art algorithms under a wide range of noise conditions implying that the algorithm is effective in practice.

1.3 Outline

In the first part of the thesis, we will review generic mathematical concepts that are used to design numerical optimization algorithms in Chapter 2. In the second part, from Chapter 3-7, we go into the details of the solution for the five problems mentioned in section 1.2. We demonstrate how a well known algorithm can easily be parallelized to solve optimization problems with constraints that arise in various computer vision applications in Chapter 3. Then, we generalize the algorithm to solve optimization problems when the objective function f is not differentiable, and analyze the convergence properties of the resulting algorithm in Chapter 4. Next, we present a way to impose statistically motivated constraints while training Deep Neural Networks in Chapter 5. Apart from continuous optimization, we show how to handle combinatorial constraints that arise in high dimensional sample selection problems to save cost in neuroscience clinical trials in Chapter 6. In the final result described in this thesis, we present a simple and robust algorithm for incorporating constraints for the blind deconvolution problem in Chapter 7. We conclude and discuss future research directions in Chapter 8.

1.4 Notations

Before we discuss the mathematical background necessary, we will list standard notations that will be followed throughout this thesis. We identify finite n dimensional vector space over the reals $\mathbb{R}$ with $\mathbb{R}^n$.

- Vectors are denoted in lower case such x, y, z, s whereas matrices and sets are denoted in upper case such as M, T, C, D, Γ.
- The $i-$th coordinate of $x \in \mathbb{R}^n$ is denoted by x_i. Similarly the entry at $i-$th row, and $j-$th column of a matrix M is denoted by M_{ij}.
- The $i-$th standard basis vector in $\mathbb{R}^n$ is denoted by e_i with the $i-$th coordinate equal to one, zero everywhere else. $\mathbf{1}$ (or 1 when there is no confusion) denotes the vector with ones in all coordinates.
- Inner product between $x, y \in \mathbb{R}^n$ is denoted by $\langle x, y \rangle$ or $x^T y := \sum_{i=1}^n x_i y_i$. $x \otimes y$ denotes the Kronecker product defined as $(x \otimes y)_{ij} = x_i y_j$. $x \circ y$ denotes the elementwise multiplication, that is, $(x \circ y)_i := x_i y_i$. $\|x\|_p$ denotes the ℓ_p norm of x.
- Given a matrix M, $M[i,:]$ or $M_{i,:}$ denotes the $i-$th row, similarly $M[:,j]$ denotes the $j-$th row. M^T denotes the transpose of M, that is the $ij-$th entry of M^T is given by M_{ji}. $\mathrm{vec}(M)$ denotes the vector obtained by stacking columns of M, one below another.
- $\mathrm{Tr}(M)$ or $\mathrm{tr}(M)$ denotes the trace of a matrix M, $\mathrm{tr}(M) := \sum_i M_{ii}$.
- Δ denotes the probability simplex in appropriate dimensions, that is, $\Delta := \{x : x_i \geqslant 0, 1^T x = 1\}$.
- A system of linear inequalities defines a polyhedron $P := \{x : Ax \leqslant b\}$. Given a polyhedron P, $\mathrm{vert}(P)$ denotes the set of vertices or extreme points of P.
- The cardinality of a finite set S is denoted by $|S|$. Given a finite set S, $\mathrm{conv}(S)$ denotes the convex hull of S defined as $\left\{ \sum_{i=1}^{|S|} \lambda_i x_i : x_i \in S, \lambda \geqslant 0, \mathbf{1}^T \lambda = 1 \right\}$.
- Given $x, y \in \Delta \subseteq \mathbb{R}^n$, $\mathrm{KL}(x\|y)$ denotes the Kullback-Leibler divergence between x and y defined as, $\mathrm{KL}(x\|y) := \sum_{i=1}^n x_i \log\left(\frac{x_i}{y_i}\right)$.

Chapter 2

Mathematical Background

In this chapter, we will briefly describe the mathematical background upon which the remainder of the thesis builds. I will present some key mathematical results on continuous optimization problems and basic convergence analysis in the first part of this chapter. Next, we will introduce the Empirical Risk Minimization (ERM) framework which is used as the modeling framework for large scale ML problems. Finally, we will briefly discuss the literature on stochastic optimization, algorithms that are commonly used to train deep networks posed in the ERM framework.

2.1 The landscape of Numerical Optimization

In this thesis, we restrict our attention to finite dimensional real valued functions $f : \Omega \to \mathbb{R}$ such that $\Omega \subseteq \mathbb{R}^n$. Optimization problems can be classified into two categories based on the presence or absence of discrete variables. That is, optimization problems where we require the decision variables x to be discrete valued are referred to as *discrete* whereas problems where x are allowed to be real valued are called *continuous.*

The flavor of algorithms that are adopted to solve continuous optimization problems tends to be slightly different from those used to solve discrete optimization problems. From the mathematical point of view, analytical techniques are used to determine the worst case complexities of continuous optimization whereas polyhedral combinatorics play a central role in analyzing discrete optimization procedures. In this chapter, we will also see concepts such as duality, and relaxations that serve as a bridge between discrete and continuous optimization. We will use the standard

undergraduate analysis notations through this chapter as used in (Rudin et al., 1964), unless otherwise mentioned.

2.1.1 Continuous Optimization

A real valued continuous function $f : \Omega \to \mathbb{R}^n$ is called *differentiable* at $x \in \Omega$ if there exists a vector $g \in \mathbb{R}^n$ (and some norm $\|\cdot\|$ defined on Ω) such that,

$$\lim_{y \to 0} \frac{f(x+y) - f(x) - g^T y}{\|y\|} = 0. \tag{2.1}$$

In the case where g that satisfies equation (2.1) exists, g is called the *gradient* of f at x. The gradient of f at x is also commonly denoted as $\nabla f(x)$. Similarly, the *directional* derivative D_f of f at x in the direction d is given by,

$$D_f(x; d) = \lim_{\epsilon \to 0} \frac{f(x + \epsilon d) - f(x)}{\epsilon}. \tag{2.2}$$

If f is differentiable and $\nabla f(x)$ is a continuous function, then f is called as a *continuously differentiable* function. If f is continuously differentiable, then the directional derivative has a linear algebraic interpretation. To see that, define the one variable "rate" function $\phi(\eta) := f(x + \eta d)$. Then, using the chain rule, we have that,

$$\phi'(\eta) = \nabla f(x + \eta d)^T d. \tag{2.3}$$

Using equation (2.3) in the definition of directional derivative (2.2), we get,

$$D_f(x; d) = \lim_{\epsilon \to 0} \frac{f(x + \epsilon d) - f(x)}{\epsilon} = \lim_{\epsilon \to 0} \frac{\phi(\epsilon) - \phi(0)}{\epsilon} = \phi'(0) = \nabla f(x)^T d, \tag{2.4}$$

where we used the continuity of the derivative in the last equality in (2.4). It turns out that directional derivatives are sufficient for optimization purposes in some cases even when the function is not differentiable.

2.1.1.1 Diagnosing Optimality

We will state the most common forms of Taylor's theorem that is used for optimization purposes.

Theorem 2.1. *Suppose f is continuously differentiable. Then we have the following:*

1. *"Mean value" form: There exists some* $\eta \in (0,1)$ *such that*

$$f(y) = f(x) + \nabla f\left(x + \eta(y-x)\right)^T (y-x). \tag{2.5}$$

2. *Assume that* f *is twice continuously differentiable.*

   a) *Then, we have the "integral" form of the gradient function as,*

$$\nabla f(y) = \nabla f(x) + \int_0^1 \nabla^2 f\left(x + \eta(y-x)\right)(y-x)\,d\eta, \text{ and}$$

   b) *There exists some* $\eta \in (0,1)$ *such that*

$$f(y) = f(x) + \nabla f(x)^T (y-x) + \frac{1}{2}(y-x)^T \nabla^2 f\left(x + \eta(y-x)\right)^T (y-x).$$

Observe that we can easily write down the integral form of equation (2.5). Using Taylor's theorem 2.1, we can recognize if a given point $x^* \in \Omega$ is optimal as shown in the following lemmas.

Lemma 2.2 (First Order Necessary Condition: 1-NC)**.** *Assume* f *is continuously differentiable. If* x^* *is a local minimizer, then* $\nabla f(x^*) = 0$*. In other words, there exists a neighborhood* B *around* x^* *such that* $f(x^*) \leqslant f(x)$ *for all* $x \in B$*.*

Definition 2.3 (Stationary Point)**.** *A stationary point is a point* $x^* \in \Omega$ *such that* $\nabla f(x^*) = 0$*, and strict minimizer is defined with a strict inequality in Lemma 2.2. A matrix* M *is positive definite if* $x^T M x > 0$ *for all* $x \neq 0$*, and positive semidefinite if* $x^T M x \geqslant 0$ *for all* $x \neq 0$*.*

Lemma 2.4. *Assume that* f *is twice continuously differentiable. Then we have the following:*

1. *Second Order Necessary Condition (2-NC): If* x^* *is a local minimizer of* f*, then* $\nabla f(x^*) = 0$ *and* $\nabla^2 f(x^*)$ *is positive semidefinite.*

2. *Second Order Sufficient Condition (2-SC): If* $\nabla f(x^*) = 0$ *and* $\nabla^2 f(x^*)$ *is positive definite, then* x^* *is a strict local minimizer.*

Even though results in Lemma 2.2, and 2.4 provide a meaningful characterization of optimal solutions, finding exact optimal solutions can be difficult in practice. Hence, we relax our goal to find approximate solutions. To that end, in addition to f, and Ω, the optimization problems are specified with a (small) accuracy parameter $\epsilon > 0$. The

quality of the output of an algorithm or the closeness of the output of an algorithm to an optimal solution can be determined by the accuracy parameter ϵ.

Remark 2.5. *In the remainder of the thesis, we will use "optimal solution" to mean an "approximately optimal" solution unless otherwise specified.*

2.1.2 Gradient Descent for Smooth functions

In this section, we will discuss the most important algorithm for optimizing continuously differentiable functions called the Gradient Descent (GD), and see how to analyze its convergence properties. As the name suggests, GD can implemented using only the gradient information of f, and is based on a simple observation using Taylor's series which we will explain now.

Given some $x \in \Omega$ such that $\nabla f(x) \neq 0$, any vector $d \in \mathbb{R}^n$ such that $d^T \nabla f(x) < 0$ is called as a *descent direction*. To see why, from equation (2.4), we know that $d^T \nabla f(x)$ is the directional derivative of f along the direction d. Hence, if η is sufficiently small positive constant, then $f(x + \eta d) < f(x)$. Indeed, since the negative gradient is a descent direction, we can define GD as the following iterative procedure:

Algorithm 1 Gradient Descent for differentiable functions

Input: Pick an arbitrary starting point $x_0 \in \Omega$, maximum number of iterations T.
for $t = 1, 2, \ldots, T$ **do**
  Update: $x_{t+1} := x_t - \eta_t \nabla f(x_t)$
end for
Output: x_T

In words, the update step in Algorithm 1 moves the current estimate of the optimal solution x_t, along the direction specified by the gradient evaluated at the current point $\nabla f(x_t)$, scaled by the scalar η_t. The scalar η_t is called the *step size* or *learning rate* depending on the context of the objective function f. Observe that if $\nabla f(x_t) = 0 \in \mathbb{R}^n$, then $x_{t+1} = x_t$, that is, GD stops making progress. More interestingly, this simple observation raises the following question: does GD output a solution with zero gradient? The answer is yes! In the following theorem, we will show that GD converges to a stationary point of f, that is, if T is sufficiently large, then $\|\nabla f(x_T)\| \approx 0$.

Theorem 2.6 (Convergence of GD). *Assume that ∇f is Lipschitz, that is, $\|\nabla f(x) - \nabla f(y)\| \leqslant \beta \|x - y\|$, and f is bounded below by $\underline{f}$. Then, the sequence of iterates $\{x_t\}$ generated by Gradient Descent Algorithm 1 satisfies $\lim_{t \to \infty} \|\nabla f(x_t)\| = 0$.*

Proof. Using Taylor's theorem or more specifically, the integral form of equation (2.5), we obtain,

$$\begin{aligned} f(x_{t+1}) - f(x_t) - \nabla f(x_t)^T (x_{t+1} - x_t) & \\ = \int_0^1 \left(\nabla f(x_t + \gamma(x_{t+1} - x_t)) - \nabla f(x_t)\right)^T (x_{t+1} - x_t) d\gamma & \\ \text{(Cauchy-Schwarz)} \leqslant \int_0^1 \|\nabla f(x_t + \gamma(x_{t+1} - x_t)) - \nabla f(x_t)\| \, \|x_{t+1} - x_t\| d\gamma & \\ (\nabla f \text{ Lipschitz}) \leqslant \int_0^1 \beta\gamma \|x_{t+1} - x_t\|^2. & \end{aligned} \tag{2.6}$$

Now using the GD update rule in Algorithm 1 with $\eta_t \equiv \eta = 1/\beta$ in inequality (2.6) we obtain,

$$f(x_{t+1}) \leqslant f(x_t) - \frac{1}{2\beta} \|\nabla f(x_t)\|^2 . \tag{2.7}$$

Summing all the inequalities in (2.7) over t, we have that,

$$\sum_{t \in \{0,1,\ldots,T\}} \|\nabla f(x_t)\| \leqslant \sqrt{2\beta \left(f(x_0) - \underline{f}\right)} < \infty \implies \lim_{T \to \infty} \|\nabla f(x_T)\| = 0. \tag{2.8}$$

□

Inequality (2.6) is often referred as the descent lemma , and we call inequality (2.7) as *per-iteration progress*. Next, we will see how to determine the cost of running the GD algorithm in order to get an ϵ-approximate solution given a fixed $\epsilon > 0$.

Oracle Complexity: The cost associated with running an iterative procedure depends on the amount of computational resources it requires in order to output an optimal solution. For our purposes, we will use the notion of an *oracle* or a black-box oracle. An oracle is an abstract computational machine that takes unit resource to compute the output. In this context, the total cost or complexity of an algorithm is simply the amount of oracle calls it needs in order to output an optimal solution. For example, a *0-th order* oracle takes a point $x \in \Omega$ and outputs the function value $f(x)$ whereas a *first order* oracle also outputs the gradient $\nabla f(x)$ at x, and so on. In general, it is not true that using higher order information (like Hessian) is always helpful to design algorithms with low (overall) complexity or cost (Roosta-Khorasani and Mahoney, 2019). In this thesis, we are mostly concerned with the first order oracle.

Oracle Complexity of GD: Observe from inequality (2.8) that $\min_{t\in\{0,1,\ldots,T\}} \|\nabla f(x_t)\|$ is $O(1/\sqrt{T})$, hence we say that GD algorithm converges at the rate $O\left(1/\sqrt{T}\right)$ or equivalently that the complexity of GD (to obtain a ϵ-approximate solution) is $O\left(1/\epsilon^2\right)$.

Theorem 2.6 is a simple convergence proof and serves as a template to design faster algorithms. We will review other facts about GD algorithm without proof to understand its theoretical properties. To begin, we state a fact about the convergence of GD which does not require the ∇f-Lipschitz assumption, see Section 3.2 in (Bertsekas and Tsitsiklis, 2003).

Fact 2.7. *Suppose that the ∇f-Lipschitz assumption in Theorem 2.6 is replaced by the following two conditions:*

1. *For every bounded set $A \subset \mathbb{R}^n$, there exists some constant K such that $\|\nabla f(x) - \nabla f(y)\| \leqslant K\|x - y\|$ for $x, y \in A$.*
2. *The set $\{x|F(x) \leqslant c\}$ is bounded for every $c \in \mathbb{R}$.*

Then we have that GD Algorithm 1 converges to a stationary point (as defined in 2.3), that is, $\lim_{t\to\infty} \nabla f(x_t) = 0$.

Fact 2.7 is particularly useful to design algorithms when β is unknown or for analyzing line search based methods. To summarize, the above results show that under some technical conditions, GD converges to a point where the gradient of the objective function vanishes. Unfortunately, (Murty and Kabadi, 1987) showed that even checking whether the output x_T of GD Algorithm 1 is locally optimal is NP-Hard. The following result shows that if we choose η_t as a constant, independent of x_t, then GD Algorithm 1 converges to a local minimizer.

Fact 2.8. *Let f be twice continuously differentiable, ∇f be $\beta-$lipschitz and $\eta_t \equiv \eta \in (0, 1/\beta)$. Let x^* be a strict saddle point of f, that is, the minimum eigenvalue of the Hessian of f at x^* is negative. Then the probability that GD Algorithm 1 converges to x^* is zero.*

In Fact 2.8, the probability is computed with respect to the distribution of the starting point x_0. For an important family of functions, it turns out that any local minimizer is also a global minimizer.

2.1.3 Gradient Descent for Convex functions

Definition 2.9 (Convexity). *A real valued function f is convex if Jensen's inequality holds:*

$$f((1-\lambda)x + \lambda y) \leqslant (1-\lambda)f(x) + \lambda f(y) \; \forall \, x, y \in \Omega, \lambda \in [0,1].$$

Similarly, we can define a convex set C as:

Definition 2.10. *A set $C \subseteq \mathbb{R}^n$ is convex if $(1-\lambda)x + \lambda y \in C \; \forall \, x, y \in C$ and $\lambda \in [0,1]$.*

We will state the following facts about convex functions and sets that will be useful throughout the thesis without proof.

Fact 2.11.

1. *If f is convex, then f is continuous in the interior of its effective domain.*
2. *Let $f_i : i \in \{1, \ldots, m\}$ be a set of m convex functions. Then $f(x) = \sum_{i=1}^{m} f_i(x)$ and $f(x) = \max_{i=1}^{m} f_i(x)$ are convex functions.*
3. *Fix $\alpha \in \mathbb{R}$. Then the α-sublevel set defined as $\{x \in \mathbb{R}^n : f(x) \leqslant \alpha\}$ of a convex function f is convex.*
4. *If f is convex, any local minimizer is a global minimizer of f. If in addition f is differentiable, then any stationary point is a global minimizer of f.*
5. *Assume f is differentiable. Then, f is convex if and only if for all $x, y \in \Omega$, we have that,*

$$f(y) \geqslant f(x) + \nabla f(x)^T (y - x). \tag{2.9}$$

It turns out that differentiability of a convex function may not be necessary for optimization purposes and can be further relaxed. To motivate this, observe that the inequality (2.9) essentially states that that convexity of f is equivalent to the property that the gradient of f at any point x provides a global lower bound for f. The set of linear functions that bound f from below is called the *subdifferential*.

Definition 2.12 (Subgradient). *A vector g is called a subgradient of f at x if*

$$f(y) \geqslant f(x) + g^T (y - x). \tag{2.10}$$

The set of all such subgradients g of f at x is called the subdifferential, denoted by $\partial f(x)$.

Subgradient generalizes the notion of gradient, as shown by the following facts from convex analysis:

Fact 2.13. *Assume that* f *is convex,* $x \in \Omega$*. Then we have the following:*

1. $\partial f(x)$ *is a closed convex set.*
2. *If* $x \in$ *int dom* f*, then* $\partial f(x)$ *is nonempty and bounded where int dom corresponds to the* interior *of the* domain *of* f*.*
3. *If* f *is differentiable, then* $\partial f(x) = \{\nabla f(x)\}$*.*
4. *Let* f_1, f_2 *be convex functions,* $\alpha_1, \alpha_2 \geqslant 0$*, and* $f(x) = \alpha_1 f_1 + \alpha_2 f_2$*. Then we have that,*

$$\partial f(x) = \alpha_1 \partial f_1(x) + \alpha_2 \partial f_2(x).$$

5. *Let* $f(x) = h(Ax + b)$ *for some convex function* h*, linear operator* $A : \mathbb{R}^n \to \mathbb{R}^m$*, and* $b \in \mathbb{R}^m$*. Then we have that,*

$$\partial f(x) = A^T h(Ax + b).$$

6. *Let* $f_i : i \in \{1, \ldots, m\}$ *be a set of* m *convex functions, and* $f(x) = \max_{i \in \{1,\ldots,m\}} f_i(x)$*. Then we have that,*

$$\partial f(x) = conv\left(\bigcup_{i \in I(x)} \partial f_i(x)\right)$$

where $I(x) = \{i : f_i(x) = f(x)\}$ *denotes the active indices at* x*, and conv*(S) *denotes the intersection of all convex supersets of* S*.*

Similar to when f is smooth, we have the following optimality condition to diagnose optimality when f is not differentiable:

Fact 2.14. x^* *minimizes* f *if and only if* $0 \in \partial f(x^*)$*.*

Using subgradients, we may easily extend the Gradient Descent Algorithm 1 to optimize nonsmooth functions, and is called *subgradient descent*, as shown in Algorithm 2. In certain ML scenarios as in Section 2.1.8, it may be only possible to approximate the gradient (or subgradient). Under various technical conditions on the quality of the approximation, it is possible to analyze the convergence behavior.

Algorithm 2 Subgradient Descent for convex functions

Input: Pick an arbitrary starting point $x_0 \in \Omega$, maximum number of iterations T.
for $t = 1, 2, \ldots, T$ **do**
  Choose a subgradient: $g_t \in \partial f(x_t)$
  Update: $x_{t+1} := x_t - \eta_t g_t$
end for
Output: x_T

In general, we need to be careful when optimizing a nonsmooth f (even when f is convex) since any subgradient is not, in general, a descent direction. To see this, let $f(x) = |x|$, then the subdifferential at the origin 0 is given by the interval $[-1, +1]$. Indeed, any nonzero $0 \neq g \in [-1, +1]$ is not a descent direction. Moreover, generalizing this example to $\mathbb{R}^n$ with $f(x) = \|x\|_1$ shows that if we were to pick a uniformly random $g \in \partial f(e_1)$ where e_1 is the first basis vector, then the probability that g is a descent direction is exponentially small in the dimension n.

Fortunately, if f is convex, we can show that negative subgradient "descends" on the distance to an optimal point. To see this, let f be convex, $f(z) < f(x), g \in \partial f(x)$, then,

$$\begin{aligned} \|x - \eta g - z\|_2^2 &= \|x - z\|_2^2 - 2\eta g^T (x - z) + \eta^2 \|g\|_2^2 \\ (\text{convexity of } f) &\leqslant \|x - z\|_2^2 - 2\eta (f(x) - f(z)) + \eta^2 \|g\|_2^2. \end{aligned} \tag{2.11}$$

Setting z to be an optimal solution x^* in inequality (2.11), we can see that,

$$0 < \eta < \frac{2 (f(x) - f(x^*))}{\|g\|_2^2} \implies \|x - \eta g - x^*\|_2^2 < \|x - x^*\|_2^2. \tag{2.12}$$

Using inequality (2.12), we can show the following convergence result for Subgradient Descent Algorithm 2:

Theorem 2.15. *Let x_t be generated by the sequence generated by Subgradient iteration 2 with $\eta_t = 1/\sqrt{t}$. Assume that $\|x_0 - x^*\| \leqslant R$, $\|g\|_2 \leqslant M$ for all $g \in \partial f(x)\ \forall\ x \in \Omega$, and $0 \in \partial f(x^*)$. Then we have that,*

$$\sum_{t \in \{0,1,\ldots,T\}} [f(x_t) - f(x^*)] \leqslant \frac{R^2}{2\sqrt{T}} + \frac{1}{2} \sum_{t \in \{0,1,\ldots,T\}} \frac{M^2}{\sqrt{T}}.$$

Theorem 2.15 implies that the convergence rate of Subgradient Descent Algorithm

2 is $O(1/\sqrt{T})$ which is essentially the same as that of Gradient Descent Algorithm 1 (as shown in Theorem 2.6).

2.1.4 Gradient Descent for Constrained Optimization

So far we have seen algorithms for optimizing functions with no *explicit constraints*. Often, we may want to solve optimization problems over a subset $C \subset \mathbb{R}^n$, $C \neq \mathbb{R}^n$. In general, we wish to solve problems of the form

$$\min_x f(x) \quad \text{subject to} \quad x \in C. \tag{2.13}$$

We will assume throughout this section that f is convex, and C is a closed convex set, not necessarily $\mathbb{R}^n$ in problem 2.13. The structure of the constraints C, in addition to f, play an important role in designing and analyzing algorithms to solve problem (2.13).

2.1.4.1 Linear Programs and Friends

Unarguably, linear $f \in \mathbb{R}^n$ with a set of m linear constraints of the form $a_i^T x \leqslant b_i \forall i = 1, \ldots, m$ or simply $Ax \leqslant b$ where $A : \mathbb{R}^n \to \mathbb{R}^m$, $b \in \mathbb{R}^n$ represents the simplest form of constrained optimization problem called a *Linear Program (LP)*. LPs can be solved *exactly* by a family of algorithms called *Simplex*. To see why this is possible, we recall that a Simplex method is based on the principle that if there exists a solution to a LP, then there is at least one vertex (or an extreme point) of the polyhedron $\{x : Ax \leqslant b\}$ that is optimal. Using an appropriate pivoting rule, a simplex algorithm essentially cycles through the vertices of the polyhedron one-by-one, making the computation of an exact solution possible (Ferris et al., 2007). Unfortunately, all the known pivoting rules can take exponential (in m or n) number of steps in finding an optimal solution, hence all known simplex methods are not polynomial time algorithms (Shamir, 1987). Having said that, an approximate solution of a LP can be computed using a different family of methods called *Interior Point Method (IPM)* in $O\left(\sqrt{\max(m, n)}\right)$ worst case complexity. In fact, IPM can be used to solve the broader class of SDPs and has polynomial time worst case complexity (Alizadeh, 1995). When f and C are convex and quadratic, problem (2.13) is called as a Quadratic Program (QP) which can be

equivalently formulated as a SDP. To see why, we note that any QP can be written as,

$$\min_x x^T Q_0 x + q_0^T x + c_0 \quad \text{s.t.} \quad x^T Q_i x + q_i^T x + c_i \leqslant 0 \; \forall \; i = 1, \ldots, m, \tag{2.14}$$

where $Q_i, i = 0, \ldots, m$ are positive semidefinite. QPs are used in ML tasks such as task plan optimization (Hadfield-Menell et al., 2016), and finding optimal trajectories for biped robots (Werner et al., 2012). Adding a scalar auxiliary variable, problem (2.14) is the same as,

$$\min_{x,\alpha} \alpha \quad \text{s.t.} \quad x^T Q_0 x + q_0^T x + c_0 - \alpha \leqslant 0, x^T Q_i x + q_i^T x + c_i \leqslant 0 \forall i = 1, \ldots, m.$$

Now since all Q_i's are positive semidefinite, we can use their Cholesky factorization to write $Q_i = M_i^T M_i$ for some matrix M_i. Then we use the Schur's complement to see that each quadratic constraint can be equivalently defined as a positive semidefinite constraint as,

$$\begin{bmatrix} I & M_i x \\ x^T M_i^T & -c_i - q_i^T x \end{bmatrix} \succeq 0 \iff x^T Q_i x + q_i^T x + c_i \leqslant 0,$$

where $\succeq$ denotes the partial ordering on the set of positive semidefinite matrices. Hence we can write the SDP formulation of QP 2.14 as,

$$\min_{x,\alpha} \alpha \text{ s.t.} \begin{bmatrix} I & M_0 x \\ x^T M_0^T & -c_0 - q_0^T x + \theta \end{bmatrix} \succeq 0, \begin{bmatrix} I & M_i x \\ x^T M_i^T & -c_i - q_i^T x \end{bmatrix} \succeq 0 \forall i = 1, \ldots, m.$$

The discussion above shows that optimizing linear/quadratic f over C defined by linear/quadratic inequalities can be handled efficiently. Now we will use these template optimization problems such as LP, and QP to solve a more general class of optimization problems with a convex objective function f. To that end, we will use an operation that is well defined for convex sets called *Euclidean Projection* onto C.

Definition 2.16 (Euclidean Projection)**.** *The euclidean projection denoted by $\Pi_C(\cdot)$ is defined as,*

$$\Pi_C(y) := \arg\min_{x \in C} \|x - y\|_2^2. \tag{2.15}$$

Using the Π_C operator (2.15), we can define the prototypical method that is used

to solve constrained optimization called Projected Gradient Descent algorithm 3.

Algorithm 3 Projected Gradient Descent (PGD) for convex f, C

Input: Pick an arbitrary starting point $x_0 \in \Omega$, maximum number of iterations T.
for $t = 1, 2, \ldots, T$ **do**
  Choose a subgradient: $g_t \in \partial f(x_t)$
  Subgradient step: $y_t := x_t - \eta_t g_t$
  Euclidean projection onto C: $x_{t+1} := \Pi_C(y_t)$
end for
Output: x_T

Intuitively, in Algorithm 3 we first update our iterate x_t using a subgradient of f, just like we did for unconstrained optimization while ignoring the constraints in C. But now, the new iterate y_t need not belong to the set C or in words, y_t need not be feasible. To correct the violation of feasibility, we simply find the nearest point to y_t in C using the projection operator Π_C. As in the case of unconstrained optimization, the optimality condition can be specified can be geometrically characterized in the following fact:

Fact 2.17. *Any point x^* minimizes a f over C if and only if there exists a subgradient $g \in \partial f(x^*)$ such that $g^T(y - x) \geqslant 0 \; \forall \, y \in C$.*

Using fact 2.17 we can show that projection is a 1-Lipschitz mapping, that is,

$$\|\Pi_C(x) - \Pi_C(y)\| \leqslant \|x - y\| \; \forall \, x, y \in \Omega. \tag{2.16}$$

It turns out that the above 1-Lipschitz property (2.16) is sufficient in order to analyze projection based descent algorithms. The following theorem characterizes the convergence rate of Algorithm 3 for nonsmooth functions:

Theorem 2.18. *Suppose f is convex and L-Lipschitz, that is, for any $x \in C$, we have that $\|g\| \leqslant L \; \forall \; g \in \partial f(x)$. Then iterates $\{x_t\}$ generated by the Projected Gradient Descent Algorithm 2 with $\eta_t = 1/\sqrt{t}$ satisfies the following inequality:*

$$f\left(\frac{1}{T}\sum_{t\in\{0,1,\ldots,T\}} x_i\right) - f(x^*) \leqslant O\left(\frac{L}{\sqrt{T}}\right).$$

Theorem 2.18 shows that the overall complexity of Algorithm 3 is the same as the unconstrained subgradient descent Algorithm 2. However, note that this result

imposes two important requirements regarding the projection operator Π_C: that Π_C can be computed (i) efficiently, and (ii) exactly. These two requirements on the projection operator can be too restrictive for practical purposes, especially when n is large. The simplest example is when C is given by linear *inequality constraints* in which exact projections are often not possible or even computing approximate solutions *per iteration* is computationally expensive (Anstreicher, 1996). More generally, in learning ML models, the objective function f can often be written as a finite sum over the (observed) data points sampled from an unknown distribution. This means that it may be possible to easily compute an *approximate* gradient of the sum f by subsampling techniques. Hence, while learning such ML models, the complexity of the overall algorithm scales linearly with the cost of computing the projection operator Π_C which is obviously undesirable.

2.1.5 Conditional Gradient (CG) Algorithm for Smooth f

Following the discussion above, the motivation for this algorithm is that the projection operator can be computationally inefficient in certain scenarios. The CG algorithm essentially replaces the quadratic objective function in the definition of Π_C (2.15) by a linear objective function, hence is often referred as *projection-free*. Intuitively, since f is smooth, we may simply use the linear approximation provided by the Taylor's series (around the current iterate) of f. CG then takes a step from the current iterate towards the minimizer of this linearized subproblem. CG algorithm to minimize a smooth f is given in Algorithm 4. We will refer to the CG sometimes as Frank Wolfe (FW) Algorithm. Observe that if $x_t \in C$, then $x_{t+1} \in C$ since x_{t+1} is simply a convex combination of x_t and y_t, both of which are in C. Clearly, we can see that CG can be

Algorithm 4 Conditional Gradient Descent for smooth f

Input: Pick an arbitrary starting point $x_0 \in C$, maximum number of iterations T.
for $t = 1, 2, \ldots, T$ **do**
  Linear Optimization (LO) call: $y_t \in \arg\min_{x \in C} f(x_t) + \nabla f(x_t)^T (x - x_t)$
  CG update: $x_{t+1} = x_t + \eta_t (y_t - x_t)$
end for
Output: x_T

applied only if C is compact (closed and bounded) since linear optimization call is well defined only when C is bounded. It turns out CG for smooth f can be shown to be as efficient as PGD algorithm 3 in the following theorem.

Theorem 2.19. *Let f be convex, ∇f be β-Lipschitz, Assume that the diameter D of C to be R, that is, $D := \sup_{x,y \in C} \|x - y\|$. Then the iterates $\{x_t\}$ generated by the CG algorithm 4 satisfies,*

$$f(x_T) - f(x^*) \leqslant \frac{2\beta D}{T+1}.$$

Proof. We prove this result since we will use it in the subsequent chapters of this thesis. Using Taylor's theorem on f we get,

$$\begin{aligned}
f(x_{t+1}) &\leqslant f(x_t) + \nabla f(x_t)^T (x_{t+1} - x_t) + \frac{\beta}{2}\|x_{t+1} - x_t\|_2^2 \\
\text{(Definition of } x_{t+1}) &\leqslant f(x_t) + \eta_t \nabla f(x_t)^T (y_t - x_t) + \frac{\beta}{2}\|\eta_t y_t - \eta_t x_t\|_2^2 \\
\text{(Definition of D)} &\leqslant f(x_t) + \eta_t \nabla f(x_t)^T (y_t - x_t) + \frac{\beta}{2}\eta_t^2 R^2 \\
\text{(Definition of } y_t) &\leqslant f(x_t) + \eta_t \nabla f(x_t)^T (x^* - x_t) + \frac{\beta}{2}\eta_t^2 R^2 \\
\text{(Convexity of f)} &\leqslant f(x_t) + \eta_t \left(f(x^*) - f(x_t)\right) + \frac{\beta}{2}\eta_t^2 R^2. \qquad (2.17)
\end{aligned}$$

Using $\eta_t = 2/(t+1)$ in equality (2.17) and an induction argument on t finishes the proof. □

There is a lot of work devoted to making the algorithm provably faster in specific cases, see (Garber and Meshi, 2016). It turns out that we can also study the convergence aspects of the CG algorithm, using a measure of linearity of the function on C.

Definition 2.20 (Curvature constant)**.** *The curvature constant of f denoted by C_f, is defined as,*

$$C_f := \sup_{x,y \in C, \eta \in [0,1]} \frac{1}{\eta^2} \left(f(y) - (y - x)^T \nabla f(x) - f(x)\right).$$

Intuitively, C_f measures how accurate the linear approximation is for a given function f on the feasible set C but only along a linear direction. We can show that this is, in fact, bounded by the diameter D of C and the Lipschitz constant β of ∇f. One version of the convergence proof is based on the Wolfe dual (Jaggi, 2011) and showing the duality gap is $\epsilon > 0$ for a sufficiently large T polynomial in $1/\epsilon$. The only constant that appears in the iteration complexity is the C_f mentioned above. We will go into the details in Chapter 4 of this thesis.

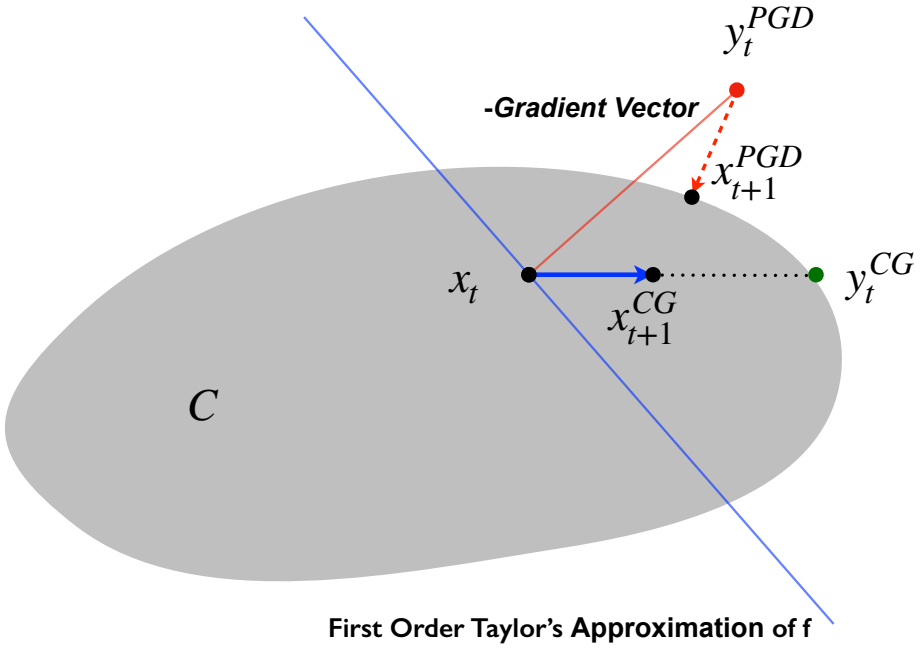


Figure 2.1: **Illustration of one iteration of PGD Algorithm 3 and CG Algorithm 4.** Here, the yellow region denotes the feasible set C, and the current iterate is x_t. Red and blue arrows denotes the execution of PGD and CG step respectively. Specifically, observe that the next iterate of PGD x_{t+1}^{PGD} is computed using the (euclidean) projection operator $\Pi_C(y_t^{PGD})$ (see Definition 2.16) while the next iterate x_{t+1}^{CG} of CG is taken as a convex combination of the current iterate x_t and the minimizer y_t^{CG} of the linear approximation of f over C (see Algorithm 4) .

2.1.6 Complexity of CG and PGD for smooth f

As mentioned above, because of its simplicity, the per iteration complexity of CG is much smaller for many popular machine learning problems compared to PGD, see (Jaggi, 2011) for an excellent discussion. We provide Figure 2.1 to visualize one iteration of PGD and CG Algorithms starting at the same location x_t. We will now see a few other ways in which both the methods compare with each other:

1. **Convergence rate** for $\Delta_t := f(x_t) - f^* \leqslant \epsilon$ accuracy (ignoring universal constants) for smooth convex f with no further assumptions:

$$\text{CG:} \quad \frac{L \cdot \text{diam}^2(C)}{t} = O(LD^2/t)$$
$$\text{PGD:} \quad \frac{L \cdot \|x_0 - x^*\|_2^2}{t} = O(LD^2/t).$$

   This means that as our analysis suggested, both CG and PGD have the same convergence guarantee.

2. **Strong Convexity.** In general, the convergence cannot be improved for CG by assuming strong convexity properties of f with a simple example whereas for PGD it can be.

3. **Affine invariance of LO oracle.** From a simple check, we can see that FW method cannot also be sped up by any change of coordinate system.

4. **Sparsity.** FW type algorithms have an inherent sparsity in its iterates x_t. For instance, suppose C is a simplex or nuclear norm ball. LO corresponds to computing a basis vector or a rank one matrix, hence after t iterations, x_t will have at most t nonzero entries or rank t. This has close connections with designing sublinear algorithms (Clarkson et al., 2012) and coresets (Har-Peled and Kushal, 2007) (approximating a set of points by a subset algorithmically). We will explore this in detail in Chapter 4.

2.1.7 Optimizing nonsmooth f using CG type algorithms

It turns out that the CG method when f is nondifferentiable is not well explored. There are two main approaches that deal with this case and point out an issue with each of them which we will address in this thesis:

1. **Gaussian smoothing.** (Hazan and Kale, 2012) proposed optimizing a smoothed version $\hat{f}_\delta$ of f by sampling. This approach is simple, and straightforward to implement. The convergence result shown is,

$$\Delta_t \leqslant Kt^{-d},$$

where K is a constant (that depends on D, L) and $d \in [0, 3/4]$. In the online or stochastic setting (where the cost functions are sampled iid from some unknown distribution), they show that we can set $d = 1/2$ giving us a regret bound of $O(1/\sqrt{t})$ which is nice. But in the offline setting, which is what we consider in this thesis in Chapter 4, in the worst case, $d = 0$ results in a constant additive error that does not go to 0 as t increases. In experiments in Chapter 4, we will investigate the run time of our method on optimizing loss functions defined by standard SVM datasets.

2. **Simulating PGD.** Another approach which is a hybrid of CG and PGD uses the so-called *Proximal* operator of a convex function f defined as,

   Definition 2.21 (Proximal Operator)**.** *The proximal operator associated with a convex function f is defined as,*

   $$prox_f(x) = \arg\min_y f(y) + \frac{1}{2}\|x - y\|_2^2.$$

   In (Argyriou et al., 2014), the authors assume that an extra (additive) function g is present in objective function with the following properties: (i) proximal operator can be computed efficiently for g, and (ii) subgradient of the convex conjugate of g can be computed efficiently. Using such a function g, they present analysis showing a $O(1/\sqrt{t})$ convergence rate. We call this as "simulating" PGD since (other than some specific cases) proximal operators in general can be thought of as projection operators by choosing g to be in the indicator function of C.

2.1.8 Empirical Risk Minimization (ERM)

So far, we have discussed optimization frameworks and algorithms that do not exploit any specific structure of the objective function f. For example, while learning ML models, it is often the case that f often takes an additive structure, that is, f can be decomposed as a sum of (possibly simpler) functions f_i. Intuitively, we can think of each f_i to be a *perturbed/corrupted* version of f, hence carries *partial* information about the true function f to be optimized. Assume that we are given a dataset $D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y)\}$, where (x_i, y_i)'s are independently and identically distributed (i.i.d.) following some underlying distribution. Then ERM aims to find a

model from certain hypothesis class H that minimizes the empirical risk,

$$\min_{\text{models}\in H} \sum_i \text{disagree}(\text{model}(x_i), y_i)$$

For a ML problem, our goal is to find a hypothesis h_s with the smallest expected risk, which is defined as

$$R[h_s] = \mathbb{E}\{f(h_s(x), y)\},$$

where $f(\cdot,\cdot)$ is a function that measures the disagreement between predicted label and true label, and the expectation is taken with respect to the underlying distribution. However, in practice, we do not know the exact form of the underlying distribution. Therefore, we use the empirical risk instead and try to find the minimizer of it. Define the empirical risk of a hypothesis h to be

$$\hat{R}_n[h] = \frac{1}{N}\sum_{i=1}^{N} f(h(x_i), y_i), \tag{2.18}$$

and define the generalization error to be

$$\epsilon_{\text{gen}}[h] = |\hat{R}_N[h] - R[h]|.$$

Minimizing the empirical risk in (2.18) to compute the best hypothesis is called as ERM,

$$\min_{h\in H} \frac{1}{n}\sum_{i=1}^{N} f(h(x_i), y_i). \tag{2.19}$$

2.1.9 Stochastic Gradient Descent (SGD) for ERM

Observe that ERM problem (2.19) can be seen as a minimization of a function with a summation over n training instances $f_i(x)$ and the decision variable x represents the parameters that we seek to learn. Hence, in order to make progress towards an optimal solution, it may be enough that on an average our gradient computations are accurate. That is, we may simply pick an index $i \in \{1,\ldots,N\}$ uniformly at random to compute the gradient direction for updating purposes. The process of picking a random direction instead of the full gradient is inherently *stochastic*, hence referred

as SGD. Usually, the convergence properties of SGD is given in terms of expectated time SGD to output an approximate solution.

Chapter 3

Making Filter Flow Practical

In this chapter, we focus on developing efficient algorithms for a family of vision tasks that can be expressed as transformation estimation problems formulated as a smooth convex optimization problem (Ravi et al., 2017). Under the assumption that the unknown transformation is bounded and smooth, we show that techniques from parallel convex optimization can be leveraged for faster optimization.

3.1 Introduction

Understanding how two or more images of the same scene are related, is a fundamental problem in computer vision. Often, the coordinate systems of the respective images are related by a camera motion whereas in other cases, the scene illumination may change, the shading may differ and/or the exposure, zoom and other parameters of the camera may be modified from one image to the other. These effects typically lead to a systemic (but *otherwise arbitrary*) transformation in the image intensities. To enable follow-up analysis, an important first step is to recover the parameters describing the relationship between the images.

While technically accurate, the above description actually covers a large class of problems with a broad stroke. In practice, instead of a common strategy, most problems in this class are addressed piecemeal, by posing it as a particular *instantiation* of the high-level "transformation estimation" objective. One makes explicit use of additional information pertaining to the *specific* problem to be solved (such as acquisition details, parameters to be estimated and application specific constraints). The representative problems in this class correspond to a number of core topics in

modern vision literature: optical flow(Revaud et al., 2015; Menze et al., 2015; Yu et al., 2015), deconvolution (Levin et al., 2009; Perrone and Favaro, 2016), non-rigid morphing (Newcombe et al., 2015), stereo (plus its variations)(Scharstein and Szeliski, 2002; Mei et al., 2013; Xue and Cai, 2016), defocus (Li et al., 2013) and so on — these are all *distinct* problems but at the high level, deal with estimating the relationship between two or more images. This compartmentalized treatment has, over the years, provided highly efficient algorithms and industry-strength implementations for numerous problems. Such solutions now drive any number of downstream turnkey applications.

Despite this diversity of highly effective and mature algorithms for each standalone problem, an interesting scientific question is the following. Given that many of these formulations seek to estimate a transformation which explains the change in image intensities over two (or more) images, can we design a *unified* formulation that is rich enough to model a *broad class of transformations* and yet offers the flexibility to precisely express the *nuances of each distinct problem* listed above? In an interesting paper a few years back, Seitz and Baker, provided precisely such a framework called *Filter Flow* (Seitz and Baker, 2009). Filter Flow models image transformations as a to-be-estimated space-variant (pixel-specific) *linear 'filter'* relating a pair of images I_1 and I_2 as,

$$I_2 = TI_1, \quad T \in \Gamma, \tag{3.1}$$

where T can be thought of as a filter (or operator) whose rows act separately on a vectorized version of the source image I_1. Observe that the inverse problem of computing the *transformation*, T, specified by the first identity is severely *under constrained*. For Model (3.1) to make sense, $T \in \Gamma$ must serve as a placeholder for the *entire* set of additional constraints on the filter which enables a unique solution that satisfies our expectations for particular problems of interest, e.g., optical flow, stereo with illumination change or affine alignment. But imagine if the problem specific requirements *can* actually be encoded as a feasibility set, Γ – then – the Filter Flow formulation offers an interesting "one stop" model where the unknown transformation we seek to estimate is *linear*. It turns out that an entire catalog of vision problems fit very nicely into this formulation: from defocus to stereo with higher order priors to optical flow with rich domain specific priors, each with its own specific feasibility set Γ. Such a formulation offers several advantages: (a) it reparametrizes traditionally

non-linear or variational formulations into an optimization problem that may be solved via just linear programming; (b) the form in (3.1) can be easily modified to incorporate additional domain specific priors which may alternatively need more significant structural modifications in an algorithm designed for a particular objective and (c) while it may not be a silver bullet for all problems expressible in this form, the corresponding solutions may provide a strong baseline and drive the development of more efficient algorithms.

From a theoretical perspective, Model (3.1) is simple and elegant. Unfortunately, a direct optimization of (3.1) is intractable for image sizes we typically encounter in practice. Running the model in a medium to large scale setting, e.g., for video sequences, is simply not possible. This may seem counter intuitive, especially since the objective and constraints are linear. So why is the model not solvable by large scale linear programming solvers? It turns out that while this approach guarantees global optimality, the constraint matrix arising from practical problem sizes, cannot be instantiated, even on a high end workstation. Even when multi-resolution pyramid schemes are adopted as a practical heuristic, the running times range from 9 to 20+ hours, depending on the type of problem and the associated constraints, a weakness acknowledged by the authors (Seitz and Baker, 2009). Furthermore, some problems require adding terms in the objective that are *non-convex*; these are solved via a series of linear programs (obtained using linear approximations at each iteration). In summary, the potential scope and applicability of this interesting formulation has not been fully realized, in large part due to its serious computational footprint. The main goal of this project is to remove this limitation and **make Filter Flow practical**.

3.1.1 Related Work

The main motivation of this project is to devise practical algorithms for Filter Flow (Seitz and Baker, 2009). In addition, we discuss several case studies in Section 3.6, specific to problems which can be modeled as Filter Flow. The literature dealing with these problems such as Optical Flow (Horn and Schunck, 1981; Baker et al., 2011), Affine Alignment (Lazebnik et al., 2004) and Stereo matching (Heo et al., 2008; Hirschmüller and Scharstein, 2009) is quite mature and is not discussed at length here to avoid digressing from the main algorithmic focus of this chapter. Before moving on, we point out that Fleet et al., (Fleet et al., 2000) also proposed linear motion models to represent varied or complex motions and many of our constructions will be applicable

to the ideas in these earlier papers.

We highlight some application domains where the filter flow model has been recently applied fairly successfully. In (Hirsch et al., 2010), the authors proposed an approach which made the filter flow construction efficient for space-variant Multiframe Blind Deconvolution. Later, (Du et al., 2011) used this formulation for computing scene depth from a stereo pair of cameras under a sequence of illumination directions. Filter Flow has been used effectively for fast removal of Non-uniform Camera Shake and the resultant blurs in (Hirsch et al., 2011). Others (Rhemann et al., 2010) have reformulated the smoothness prior in Filter Flow, to make it suitable in various applications including alpha matting. These methods demonstrate that the applicability of Filter Flow to a set of diverse problems is possible but has often involved disparate solution schemes. We believe that once the computational challenges are resolved, filter flow approaches will be more widely adopted in vision.

Our Results. Our earlier discussion suggests that commercial Linear Programming (LP) solvers are not well suited for solving (3.1). However, we find that in most instantiations of Filter Flow (in the context of specific vision problems), the problem has significant structure that can be exploited via specialized optimization schemes. In particular, we see that for each of the five *case studies* covering problems such as affine alignment and optical flow described in (Seitz and Baker, 2009), a *nearly equivalent* reformulation allows the applicability of numerical optimization schemes that cuts down the running time from tens of hours to several minutes on a standard workstation, *without making use of any heuristic strategies*. By nearly equivalent, we mean that the problems have the *same optimal solution* up to chosen tolerance. On the modeling side, we propose a new convex term that encourages sparsity which is called the *compactness* term, an ℓ_2 data term and a valid inequality that reduces the search space of the parameters. On the algorithmic side, with some more manipulation, the problems reveal additional structure appropriate for massively parallel lock-free implementations. To that end, we propose an efficient asynchronous parallel algorithm that solves our formulation 30 times faster than the previous model and are empirically competitive to the state of the art solvers both quantitatively and quantitatively. A detailed description of these properties and the corresponding algorithms with convergence guarantees is our main contribution.

Notations. We use the standard notations as specified in Section 1.4. We use e_i's to represent the standard basis vectors and $\mathbf{1}$ denotes the vector of all 1's. Also, Δ

gives the probability simplex in appropriate dimensions. Small alphabets represent vectors over $\mathbb{R}$; upper-case letters represent linear maps between vector spaces over $\mathbb{R}$.

3.2 A brief overview of Filter Flow

We briefly review the key components of Filter Flow Seitz and Baker (2009) to set up our discussion. Let I_1 and I_2 be a pair of images for which Filter Flow is to be computed. In our formulation, I_1 and I_2 can be assumed to be vectors in $\mathbb{R}^n$, where n is the total number of pixels in image I_1 (or I_2). Then filter flow can be formulated as the following optimization problem[1],

$$\min_{T \in \mathbb{R}^{n \times n}} ||TI_1 - I_2||_2^2.$$

Equivalently, the ith entry of TI_1 is the linear combination of intensities of I_1. The authors in Seitz and Baker (2009) further decompose T as $T = MK$ where M is a motion matrix and K is a kernel matrix. M^{-1} encodes the motion from I_2 to I_1, so we can write the objective as $||MI_2 - KI_1||_2^2$. In particular, in the case of pure motion, K is the identity matrix. Our construction can be employed for a nontrivial K but we assume that K is the identity matrix to simplify our presentation and without loss of generality swap I_2 and I_1. Since this formulation is under constrained, Seitz and Baker (2009) proposes some natural constraints on M, described below.

1. **Non-Negativity.** Negative coefficients are meaningless; so one imposes the constraint that $M \geqslant 0$.

2. **Row Simplex Constraint.** We also require that, the sum of coefficients in each row of M to be 1, i.e., each pixel in I_2 is a convex combination of pixels in I_1. In addition, we define a neighborhood of each pixel i as $\mathcal{N}(i)$ and ensure that pixels outside this neighborhood do not participate in the convex combination.

[1] Via personal communication Seitz and Baker (2009), we know that the norm was chosen to allow applicability of LP solvers and not central to the model otherwise.

Hence, we can write the optimization problem that we seek to solve as,

$$\begin{aligned} &\min_{M \succcurlyeq 0} \|MI_1 - I_2\|_2^2 \\ &\text{s.t.} \sum_{j \in \mathcal{N}(i)} M_{ij} = 1 \; \forall i = 1 \text{ to } n, \\ &\quad \sum_{j \notin \mathcal{N}(i)} M_{ij} = 0 \; \forall i = 1 \text{ to } n. \end{aligned} \tag{3.2}$$

While the above description serves as the basic template of the Filter Flow model, various additional constraints and terms are added in problem (3.2) to model the underlying computer vision problem of interest. Specific examples include: (i) for the Affine alignment problem, the requirement that all pixels in I_1 are transformed by an affine matrix A (the so-called GLOBA-M constraints in Seitz and Baker (2009)); (ii) for optical flow, we encourage smoothness on the affine motion of neighboring pixels; (iii) for precise integer flow, terms encouraging sparsity are introduced; (iv) for stereo, we require smoothness between the rows of M. Specific details of these terms will be described later in case studies.

3.3 Algorithmic Reformulations

Our basic hypothesis is that reformulating the filter flow problem will enable us to design practical algorithms with global convergence guarantees. Unfortunately, additional terms/constraints (described in the previous section) that are used in specific formulations of standard computer vision problems, make this task non-trivial. The reason is two-fold: (i) *non-convexity* in some of the decision variables and, (ii) even when the optimization problem is convex, the additional terms lead to *composite functions or constraints*. For example, in imposing affine smoothness (used in optical flow) we add the term $\sum_i \sum_{j \in \mathcal{N}(i)} \|A_i - A_j\|_2^2$ in the objective, where A_i is the affine motion associated with pixel i. Such terms make it hard to optimize the problem using standard off the shelf solvers, since it requires multiple passes over the data to compute the gradient. Our goal with the reformulations is to mitigate such issues while also preserving the overall behavior of the filter flow model.

Next, we will identify a few high-level issues that make the optimization challenging for the constraints mentioned in Seitz and Baker (2009) and offer alternatives. Our

line of attack (or workflow) is as follows. We will first address the main (computationally) problematic pieces of the Filter Flow model and provide tractable reformulations for each. Our hope will be that these reformulations will satisfy "nicer" technical conditions that will guide the choice of the overall optimization scheme. We will then describe how this scheme can be further specialized by exploiting the problem structure for particular computer vision problems (case studies). The objective will be that each specific instantiation should be implementable just by *modifying a few lines of code* of the overall optimization and still preserve efficiency benefits.

3.3.1 Reformulating Compactness

Ideally, we want to find a correspondence between the pair of images where each pixel in I_1 is mapped exactly to a single pixel in I_2. To ensure sparsity, Seitz and Baker (2009) proposed a *compactness* term, defined as,

$$\sum_{i} \sum_{j \in \mathcal{N}(i)} M_{ij} \|(j-i) - \sum_{ij} M_{ij}(j-i)\|_2^2$$

But the *compactness* term makes the objective function nonconvex (in fact, concave), therefore, a linearization approach had to be used in Seitz and Baker (2009). Initially the compactness term is set to zero and the optical flow is computed *without* the compactness term. After the first iteration, a linear approximation of the *compactness* function is used and the corresponding linear program (LP) is solved. This requires solving a huge LP problem multiple times (even though empirically the number of iterations used was only $3-4$). Secondly, the solver needs to do a full pass of the data (or image) in order to compute the linear approximation of compactness, which is expensive.

A Convex Compactness term? To successfully replace the *compactness* term with a more efficient substitute, we first consider some useful properties that such a term needs to encode the following behavior: 1) as the regularization parameter increases, it should tend towards keeping a single non-zero entry in each row, 2) the value of the function and its derivative should be efficiently computable and, 3) it should be easy to optimize. To this end, the ℓ_1 norm is a natural choice to get sparse solutions. But it turns out that the ℓ_1 norm is constant ($= 1$) on the feasible set of our problem, so it is not applicable.

Observation 1. *Constraints on the matrix* M *imply that* M *is a row stochastic matrix.*

Using Observation 1, if we consider each row of M to be a probability distribution on n variables, then we need the optimal probability distribution to have small support, that is, we want our compactness term to behave like a linear combination of the fewest number of delta functions. In Pilanci et al. (2012), the authors showed that a relaxed version of this problem can be solved by n second order cone programs in parallel. Later, Carli et al. (2013) provided a convex formulation that encourages sparsity on the simplex using Observation 1, and showed that it is more robust. Using these ideas, we can write our problem as,

$$\min_{M \geqslant 0} \|MI_1 - I_2\|_2^2 + \lambda_1 \sum_{i=1}^{n} \|M_{[:,i]}\|_2$$
$$\text{s.t.} \sum_{j \in \mathcal{N}(i)} M_{ij} = 1, \sum_{j \notin \mathcal{N}(i)} M_{ij} = 0, \ \forall\, i$$

where $\lambda_1 > 0$ is fixed and $M_{[:,i]}$ denotes the $i-$the column of M and λ_1 is the regularization parameter. Intuitively, the penalty is a group lasso type penalty with the groups given by the columns of M. Therefore, this term encourages that there are few nonzero entries along the columns and together with the sum to one constraint on the rows makes the rows sparse, achieving the desired property.

3.3.2 Reformulating Affine Smoothness

Ensuring smoothness of the transformation across the pixels is important for several problems such as Optical Flow. Affine smoothness terms, in such instances were shown in Seitz and Baker (2009) to outperform other second order smoothness terms and yield a smoother flow field in practice. To encourage affine smoothness, we define an explicit 6 parameter affine transformation at each pixel i, denoted by $A_i \in \mathbb{R}^{2\times 3}$. Intuitively, A_i captures the motion of the centroid of the filter at pixel i. To do so, Seitz and Baker (2009) imposes a hard "LOCA-M" constraint (set the flow equal to the centroid). Instead, we use the dual form and add the terms in the objective function ($\lambda_2 > 0$), that is,

$$\lambda_2 \sum_i \|A_i \mathbf{i} - \bar{M}_i\|_2^2$$

where $\bar{M}_i \in \mathbb{R}^2$ is the centroid of filter at pixel i and $\mathbf{i} \in \mathbb{R}^3$ is the pixel i in homogeneous coordinates with third entry equal to 1. We fix the value of λ_2 whenever approximate solutions are enough. We treat them as a quadratic penalty if more accurate solutions are required which guarantees (see Nocedal and Wright (2006)) that the *equality constraint is satisfied when our algorithm terminates, exactly as desired in Seitz and Baker (2009)*. Finally, to get a smooth flow, the following term is added to the objective function ($\lambda_3 > 0$),

$$\lambda_3 \sum_i \sum_{j \in \mathcal{N}(i)} \|A_i - A_j\|_2^2 \tag{3.3}$$

Now, we give a result showing an interesting property of the affine transformation matrix and is relevant in informing the choice of algorithm in the next section.

Lemma 3.1. $\|A_i\|_F^2 \leqslant C$ *is a valid inequality for a sufficiently large* $C > 0$.

Proof. Let $\mathbf{i} \in \mathbb{R}^3$ denote pixel i in homogeneous coordinates. From Lagrange multiplier theory, there exists a $\lambda_2 > 0$ such that terms in the objective function $\lambda_2 \|A_i \mathbf{i} - \bar{M}_i\|_2^2$ is equivalent to adding the following constraints (used in Seitz and Baker (2009)),

$$A_i \mathbf{i} = \bar{M}_i \ \forall i$$

Denote a as the first row of A_i, b as $\mathbf{i}$ and c is the first coordinate of $\bar{M}_i$. We need to show that a is bounded. Now $a^t b = \|a\|\|b\| \cos\theta = c$, since b and c are bounded, it follows that optimal $\|a\|_2^2$ is finite. This gives the result. □

Intuitively, this lemma says that the magnitude of each individual element of A_i's is bounded (with C chosen appropriately). Observe that $\mathbf{i} \leqslant 1$ (coordinatewise) and the restriction of the movement of a pixel to be within the filter implies that $\bar{M}_i$ is bounded by the neighborhood size giving us the *tightest* choice for C. More importantly, we immediately have the following corollary:

Corollary 3.2. *The minimizers of objective function in* (3.3) *is contained in a (convex) compact set.*

In words, Corollary 3.2 shows that we have not compromised any theoretical property by adding this inequality.

3.3.3 Which solver should we use?

One of the key differences between the model in Seitz and Baker (2009) and our approach is that the we replaced the ℓ_1 norm with *special type of compactness encouraging* norm making the objective a smooth convex function. In this section, we will see how this change of norm together with the above lemma enables leveraging recently developed convex optimization techniques that can speed up the optimization time by several orders of magnitude, assuming that the case study specific terms can also be manipulated and reformulated to be amenable to our optimization scheme.

We now discuss how two key properties viz., *approximation* and *randomization* will guide the choice of an appropriate optimization scheme. We motivate these choices next in the context of our model.

1. **Approximation.** Consider the problem of solving a finite dimensional optimization problem over a compact convex set. As we saw in Chapter 2, such problems are often solved using projected gradient methods (over multiple iterations), which involves projection of the objective at each step on the feasible set. This requires optimizing a quadratic function on the feasible set. These algorithms allow large changes in the working set at each iteration unlike active-set methods and therefore can be applied to large size problems. The primary disadvantage, however, is that even when the feasible set is very simple such as *probability simplex*, ellipsoids, ℓ_1 *norm ball*, the corresponding projection subproblem is nontrivial. Since these constraints are common in vision problems, we use the *Conditional Gradient Methods (CGM)* to solve our Filter Flow model. Recently, CGMs have performed well in solving many machine learning problems including SVMs and nuclear norm regularized problems, see Harchaoui et al. (2014); Jaggi (2011) though have only been sparingly used in vision. Indeed, the method thrives in practice when the following **two properties** are satisfied: i) feasible set is compact convex; and ii) optimizing a linear function over the feasible set is easy.

2. **Randomization.** As mentioned earlier, the double summation terms in (3.3) is a computational bottleneck, since it needs to make a full pass over the data to compute the gradient. It has been shown Bertsekas; Recht et al. (2011) that approximate first order information can be used to solve such large scale optimization problems, referred to as *randomized or stochastic algorithms*. These

methods are much faster than the traditional algorithms and are often provably robust to noise Hardt et al. (2015).

Our strategy is to combine Conditional Gradient methods with randomization to develop efficient *Randomized Block Coordinate Conditional Gradient* Bredies et al. (2009); Bonesky et al. (2007) algorithm (RBC-CGM) with strong sparsity and convergence guarantees to solve standard Filter Flow formulations. We show how RBC-CGMs can be efficiently used in tandem with powerful distributed convex optimization schemes resulting in practical algorithms for these problems.

Randomized Block Coordinate CGM (RBC-CGM). We propose an asynchronous randomized block coordinate descent variant of the classical CGM for our purposes.

The simplest form of our algorithm is outlined in Algorith 5. The reformulations made in section 3.3 have resulted in simple subproblems (# shaded in Algorithm 5) as desired. To compute s_d, we find the index corresponding to the minimum element of g_d and set it to 1 leaving all others to be 0 and $s_A = -Cg_A/\|g_A\|$ which requires the computation of a norm. This makes our algorithm very simple to implement and efficient.

Algorithm 5 RBC-CGM

while convergence **do**
  Pick an arbitrary pixel i.
  for $t = 0, 1, 2, \cdots, T$ **do**
    $g_d \leftarrow \nabla_{M_{[i,:]}} f$, $g_A \leftarrow \nabla_{A_i} f$

$$s_d, s_A \leftarrow \arg\min_{s,q} s^t g_d, q^t g_A \quad \text{s.t.} \quad s \in \Delta, \|q\|_F^2 \leqslant C \quad (\#)$$

    $[M_{[i,:]}; A_i] \leftarrow (1-\gamma_t)[M_{[i,:]}; A_i] + \gamma_t [s_d; s_A]$
  end for
end while

3.4 Analysis of Algorithm and Convergence

We now discuss some technical properties of the algorithm presented above.

Space Complexity and Iteration Bounds. If a pixel i is accessed t times, it is clear that the number of nonzero entries in $T_{[i,:]}$ is at most t. In other words, after a few

iterations, it is possible to obtain a reasonable estimate of the flow of the pixel i. Moreover, the neighborhood sizes in general are small relative to the size of the image which implies that after a few iterations it identifies the regions of the image to which the current pixels are moved to. Empirically, we can stop after 100 epochs, (i.e., when each pixel is accessed 100 times by the processor) if the filter size is $[-10, 10]^2$. Second, we can easily estimate the number of iterations needed before starting the algorithm for a fixed amount of memory.

We will now prove a lemma that establishes a block-descent property for this algorithm which can be used to trivially parallelize the algorithm.

Lemma 3.3. *Denote the objective as* $f(y) : \mathbb{R}^n \to \mathbb{R}$ *where* f *is convex, smooth and that* y *is partitioned into* $\mathcal{J} = \{1, ..., J\}$ *blocks, i.e.,* $y = [y_1, y_2, ..., y_J]$ *such that* $y_i \in Y_j$*, then we have that* $d_i^T \nabla_i f(y^i(t)) \leqslant -C' \|d_i(t)\|_2^2$*, where* d_i *is the direction of update i.e.,* $y_i(t+1) = y_i(t) + \gamma_t d_i(t)$ *and* $C' > 0$ *is a constant.*

Proof. Rewriting the update rule of the algorithm, we have that $y_i(t+1) = (1 - \gamma_t) y_i(t) + \gamma_t s_i(t) = y_i(t) + \gamma_t (s_i(t) - y_i(t))$. Hence we have that $d_i(t) = s_i(t) - y_i(t)$ where $s_i(t) \in \arg\min_s s^T \nabla f(y_i(t))$. We relax the containment operator to a equality since it does not affect the rest of the proof. Now,

$$
\begin{aligned}
d_i(t)^T \nabla f(y_i(t)) &= \frac{1}{\gamma_t} (s_i(t) - y_i(t))^T \nabla f(y_i(t)) \\
&= \frac{1}{\gamma_t} \left(s_i(t)^T \nabla f(y_i(t)) - y_i(t)^T \nabla f(y_i(t)) \right) \\
&\leqslant \frac{1}{\gamma_t} s_i(t)^T \nabla f(y_i(t)) < 0
\end{aligned}
$$

The first inequality is due to the definition of $s_i(t)$, that is, it is the minimizer of the linear subproblem problem (#) in algorithm 2 whereas the strictly inequality is due to the fact that $s_i(t)$ is a feasible direction and $\gamma_t > 0$. Hence we can choose C' accordingly to get the desired result. □

Predetermined step sizes allow for easy parallelization. Using the above lemma and proposition 5.1 in Bertsekas and Tsitsiklis (1989), we can deploy an asynchronous algorithm assuming that the time delay between the updates of processors is bounded. Of course, this does not prove that each update decreases the objective function maintaining feasibility. In fact, this is rarely true in conditional gradient type methods except if we use line search methods to compute step sizes γ_t that satisfy Armijo

condition. But this defeats the purpose of asynchronous methods because each processor will take arbitrarily long to do a single update thus affecting the convergence rate. Moreover, constant step size policies usually depend on parameters of the objective function and constraints that are hard to compute *a priori*. Fortunately, in the convergence proofs of this method in Section 3.5, and later chapter 4, we show that a particular choice of stepsize sequence determined *a priori guarantees* convergence (see Jaggi (2011)) and only depends on the iteration number making the algorithm naturally easy to parallelize Bertsekas and Tsitsiklis (1989). It is useful to see that the convergence is established using the duality gap principle and not just using the primal optimization problem. The above lemma shows the correctness of our algorithm, that is, any limit point of the sequence generated by our algorithm converges to the optimal solution at the rate equal to its sequential version, that is, $\mathcal{O}(1/\sqrt{N})$. For explicit convergence rates, see Wang et al. (2016). Many aspects that are considered in Wang et al. (2016) like collisions, delayed updates do not affect the problems considered here since the delay between individual workers is negligible. So, our proof is much simpler. Before we see specific case studies, we will provide the typical values of C_f in vision problems.

3.5 C_f Calculations

We will first mention few basic convergence results for the deterministic CGM that were proved by others for the sake of completeness.
Consider the following constrained finite dimensional optimization problem,

$$\min_x f(x) \quad \text{s.t.} \quad x \in \mathcal{C}$$

where f is a differentiable convex function and $\mathcal{C}$ is a compact convex set. We recall the key convergence result from Chapter 2 regarding the CGM.

Theorem 3.4. *The deterministic CGM (i.e., Algorithm 4) in Chapter 2 with* $\gamma_t = \frac{2}{2+t}$ *satisfies,*

$$f(x_t) - f(x_*) \leqslant \frac{4C_f}{t+1}$$

Here C_f is a geometric quantity called as the *curvature constant* that depends on the objective function f, feasible set $\mathcal{C}$ and is defined as,

$$C_f := \sup_{x,s\in\mathcal{C},\gamma\in[0,1]} \frac{1}{\gamma^2}\left(f\left(x+\gamma\left(s-x\right)\right)-f(x)-\gamma\left\langle s-x,\nabla f(x)\right\rangle\right)$$

Intuitively, C_f measures how close the function is to the linear approximation at $x \in \mathcal{C}$. If C_f is very high, it takes the algorithm many more iterations to converge to a predetermined $\epsilon-$accuracy. For nonsmooth functions, the definition of C_f is tricky, usually the nonsmooth function is approximated by a smooth function (surrogate) before running the algorithm and the analysis is done on the surrogate function. Since derivative is a linear operator, trivially we see that,

$$C_{\sum f_i} = \sum_i C_{f_i}.$$

Our objective function is the sum of two types of terms viz data term and smoothness term where each of these two is a sum of terms summed over the pixels. Hence we show how to calculate for one such term and then we can use equation 3.5 to finish.

To that end, let $f_i := g_i + h_i$ where g_i is the data term associated with pixel i and h_i is the smoothness associated with pixel i. We now show how to calculate C_{f_i} by calculating C_{g_i} and C_{h_i} separately. Since g_i is twice differentiable with $\nabla^2 g_i = I_1 I_1^T$, using Taylor's series approximation and the fact that $\gamma \in [0,1]$, we have that,

$$\begin{aligned} C_{g_i} \leqslant \sup_{x,s\in\mathcal{C}} \|I_1^T(x-s)\|_2^2 &\leqslant \sup_{x,s\in\mathcal{C}} \|I_1\|_2^2\|x-s\|_2^2 \\ &= \|I_1\|_2^2 \sup_{x,s\in\mathcal{C}} \|x-s\|_2^2 \end{aligned}$$

where the inequality is due to Cauchy-Schwarz inequality. Supremum is nothing but the squared diameter of $\mathcal{C}$, using the fact the $\mathcal{C}$ for the data term is just the probability simplex, we have that,

$$C_{g_i} \leqslant \sqrt{2}\|I_1\|_2^2$$

We can make the bound tighter since the filter size is not the whole image, $C_{g_i} \leqslant$

$\sqrt{2}||I_1^{\mathcal{N}^i}||_2^2$ where $I_1^{\mathcal{N}^i}$ denotes the coordinates of I_1 that are in the neighborhood $\mathcal{N}^i$ of pixel i.
Similarly, using the fact that $\nabla^2 h_i = |\mathcal{N}^i| I$ where $|\cdot|$ is the cardinality function and I is the identity matrix, we can compute that,

$$C_{h_i} \leqslant \sqrt{6|\mathcal{N}^i|}$$

where the number 6 appears because of the equivalence of $p-$norms in $\mathbb{R}^n$.

3.6 Case Studies

We study three specific problems, that immediately benefit from the fast filter flow formulation. We also discuss parallel solvers, if applicable.

1. **Affine Alignment.** The Affine Alignment problem deals with the task of finding global affine alignment between a pair of images. It can be formulated as

$$\begin{aligned} &\min_{M,\mathbf{A}} ||MI_1 - I_2||_2^2 \\ &\text{s.t. } M[i,:] \in \Delta, ||\mathbf{A}||_F^2 \leqslant C, \mathbf{A}\mathbf{i} = \bar{M}_i \ \forall i \end{aligned} \tag{3.4}$$

   where $\mathbf{A} \in \mathbb{R}^{2\times 3}$ is the global affine matrix capturing the affine warp between I_1 and I_2, Δ is the unit simplex, and $M[i,:]$ is the $i-$th column of M. Let A_i be the affine matrix for each pixel i. Then, we can write an equivalent form of (3.4) as,

$$\begin{aligned} &\min_{M,\mathbf{A}} ||MI_1 - I_2||_2^2 \\ &\text{s.t. } M[i,:] \in \Delta, A_i\mathbf{i} = \bar{M}_i, A_i = \mathbf{A}, ||A_i||_F^2 \leqslant C \ \forall i \end{aligned}$$

   After dualizing the equality constraints $A_i = \mathbf{A}$, we can write the optimization problem as ($\lambda > 0$),

$$\begin{aligned} &\min_{M,\mathbf{A}} ||MI_1 - I_2||_2^2 + \lambda \sum_i ||A_i - \mathbf{A}||_F^2 \\ &\text{s.t. } M[i,:] \in \Delta, A_i\mathbf{i} = \bar{M}_i, ||A_i||_F^2 \leqslant C \ \forall i \end{aligned} \tag{3.5}$$

   Note that this problem satisfies the two properties mentioned in section 3.3.3.

Parallelization. The model (3.5) can be easily parallelized as follows. Each worker picks a pixel i, solves the corresponding optimization problem and updates A_i. Observe that we do not explicitly impose that $\|A\|_F^2 \leqslant C$ in the model. After all the workers update A_i, A can be updated as,

$$A \leftarrow \arg\min_{A} \sum_i \|A_i - A\|_F^2$$

But the above optimization problem simply computes the mean of all A_i's which can be done in time linear in the number of pixels. We use incremental gradient descent method with $0 < \alpha < 1$ as the dual step size to update λ, see Bertsekas, $\lambda \leftarrow \lambda + \alpha \cdot (\sum_i \|A_i - A\|_F^2)$.

2. **Optical Flow.** Putting together the objective and constraints from Section (3.3) pertaining to optical flow, we can write the optimization problem as follows:

$$\begin{aligned}
&\min_{M \geqslant 0, A_i} \|MI_1 - I_2\|_2^2 + \lambda_1 \sum_{i=1}^{n} \|M_{[:,i]}\|_2 \\
&+ \sum_i \lambda_2^i \|A_i \mathbf{i} - \bar{M}_i\|_2^2 + \lambda_3 \sum_i \sum_{j \in \mathcal{N}(i)} \|A_i - A_j\|_2^2 \\
&\text{s.t.} \sum_{j \in \mathcal{N}(i)} M_{ij} = 1, \sum_{j \notin \mathcal{N}(i)} M_{ij} = 0, \|A_i\|_F^2 \leqslant C \ \forall i
\end{aligned}$$

Note that λ_1 and λ_3 are parameters for the optimization problem, therefore user specified constants. The main difference in this model from the affine alignment problem is that we use a different dual variable λ_2^i for each pixel. This is often useful since optical flow is computed using a pyramid approach, so we get a good initialization of A_i's locally. Again, we see that this problem satisfies the two properties mentioned in section 3.3.3.

Parallelization. Each worker solves the following problem,

$$\begin{aligned}
&\min_{M_{[i,:]}, A_i} (M_{[i,:]}^t I_1 - e_i^t I_2)^2 + \lambda_2^i \|A_i \mathbf{i} - \bar{M}_i\|_2^2 \\
&+ \sum_{j \in \mathcal{N}(i)} \|A_i - A_j\|_2^2 + \lambda_1 \sum_{i=1}^{n} \|M_{[:,i]}\|_2 \\
&\text{s.t. } M_{[i,:]} \in \Delta, \ \|A_i\|_F^2 \leqslant C
\end{aligned}$$

The optimization problems and the λ_2^i can be updated and stored locally in each worker. This gives us a lock free asynchronous parallel algorithm. After each worker finishes the above subproblem, λ_2^i are updated as,

$$\lambda_2^i \leftarrow \lambda_2^i + \alpha \cdot \left(||A_i \mathbf{i} - \bar{M}_i||_2^2 \right).$$

3. **Stereo.** Stereo matching problem is formulated using a local smoothness model instead of global affine smoothness as follows:

$$\min_{M \geqslant 0} ||MI_1 - I_2||_2^2 + \lambda_1 \sum_{i=1}^{n} ||M_{[:,i]}||_2 + \lambda_3 \sum_{i,j} ||M_i - M_j||_2^2$$
$$\text{s.t.} \sum_{j \in \mathcal{N}(i)} M_{ij} = 1, \sum_{j \notin \mathcal{N}(i)} M_{ij} = 0, \ \forall i$$

Updates here are a special case of updates for the optical flow problem, so the same parallelization scheme applies.

3.7 Experiments

We present experimental results on three case studies introduced in Section 3.6. Our goal is to evaluate (a) the degree of runtime improvements of our algorithm over alternatives which uses no reformulation strategies; (b) whether the reformulated Filter Flow model when solved to optimality can, in fact, yield results competitive with *dedicated* algorithms developed *specifically for each case study*; and (c) whether the overall scheme is efficient enough to enable rapid prototyping for new problems in vision that fit into the model in (3.2). We discuss these issues next.

Initialization. We use the Lucas Kanade algorithm to get an initial estimate of the flow, which works for most settings.

Step Size. The choice of the stepsize γ_t determines convergence. While line search can be used, it may be sub-optimal for the (partially) asynchronous aspect of our algorithm (e.g., time delay between processors). For our parallelized version, each processor uses its own iterations count.

3.7.1 κ for initialization

We use the strategy proposed in Freund and Grigas (2013) by setting $\gamma_t = \frac{2}{\kappa+t+2}$ where $\kappa > 0$ is a constant that depends on the objective.6 κ is defined as (see Freund and Grigas (2013) for details),

$$\kappa = \frac{2C_f}{|B_1 - f(x_1)|}$$

C_f is the quantity computed in the previous, $|B_1 - f(x_1)|$ is the Wolfe duality gap and since f is convex, lower bound B_1 can be easily computed using the subgradient inequality (2.10) as $f(x_1) + \nabla f(x_1)^T(s_1 - x_1)$. In order to get a good estimate of B_1, we randomly initialize $x_1 \in \mathcal{C}$. In our cases we the duality gap takes the form of the sum of duality gaps between each pixel, hence, we randomly pick a feasible point on the probability simplex and initialize T. For problems that require LOCA-M constraints (mentioned in the main paper), we solve a linear program containing 6 variables for each pixel and get A_i such that $A_i \mathbf{i} = \bar{M}_i$. That is,

$$\min_{A} \quad 0 \quad \text{s.t.} \quad A\mathbf{i} = \bar{M}, -C \leqslant A \leqslant C$$

Note that the above optimization problem is always feasible from lemma 3.1 for some $C > 0$ and we *do not* require a linear programming solver to solve problem 3.7.1.

Implementation. We used a 8-core 3.60GHz processor machine with 12GB RAM, and Intel TBB within C++ for task parallelism. We fix $\lambda_3 = 0.005$, $\lambda_1, \lambda_2 = 1$ for *all problems and all datasets*. *No other parameter tuning was performed.*

Note that evaluating Filter Flow Seitz and Baker (2009) on high resolution images is problematic because of the cost of solving the corresponding LP. Therefore, we use a state of the art optical flow method Revaud et al. (2015) for comparison. For low resolution images, the results of our method and Filter Flow were qualitatively and quantitatively similar.

3.7.2 Optical Flow

We start with the Optical Flow problem since from a Filter Flow perspective, it is computationally the most demanding Seitz and Baker (2009). We used two standard optical flow datasets: MPI Sintel Butler et al. (2012) and Middlebury Baker et al. (2011).

Figure 3.1: Optical flow results on the MPI Sintel dataset. Columns 1 to 3 (5 to 7) show the ground truth, our result and the Epicflow result respectively. Column 4 shows the error map. AEE's of column 4 are 1.08, 1.15, 1.03 for rows 1 to 3. **Our results are marked in red.**

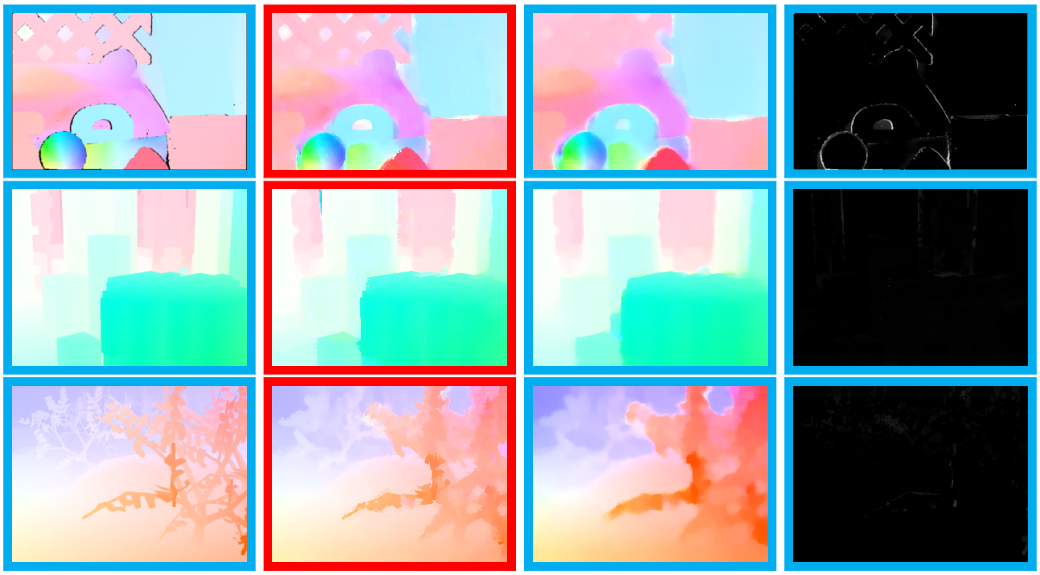


Figure 3.2: Optical flow results on the Middlebury dataset. Columns 1 to 3 (5 to 7) show the ground truth, our result and the Epicflow result respectively. Column 4 shows the error map. AEE's for column 4 are 0.06, 0.042, 0.033 for rows 1 to 3. **Our results are marked in red.**

MPI Sintel is a large displacement dataset whereas Middlebury has more nonlinear movements. MPI Sintel consists of two versions of sequences, clean and final with the same ground truth. To show the performance (both qualitative and runtime) of the "standard" Filter Flow formulation in Seitz and Baker (2009), the model was not augmented with additional constraints to handle occlusion, blur and lighting effects

explicitly. Therefore, we report results for our method and Epicflow Revaud et al. (2015) on the clean sequences (note that additional terms within Γ can incorporate these case specific constraints). Representative qualitative results are shown in Fig. 3.1 for MPI Sintel and Middlebury datasets. Results on the test set of MPI Sintel is shown in Fig. 3.3. The results in Fig. 3.1 strongly suggest that qualitatively, our results are clearly comparable to those obtained from Epicflow. In MPI Sintel (Fig. 3.1 (top row)), our results show better consistency with the ground truth (wedge on the left, small flows in the hair region). The error discrepancy map in column 4 is nearly all black. In Fig. 3.1 (row 2), our solution is able to accurately recover the flow in the yellow region although the estimation in the head region is more blurred. In Fig. 3.1 (row 3), the results from both methods are identical. The Average Endpoint Error (AEE), calculated as Frobenius norm over the number of pixels, was in the range $[0.03, 0.06]$ and $[1.02, 1.2]$ for Middlebury and MPI Sintel datasets respectively for the non-occluded pixels. Quantitatively, this is competitive with recent optical flow papers, that report results on these datasets. In most other sequences, we see a similar overall behavior where our solution has good consistency with the ground truth. This is particularly encouraging because other than the constraints described in Section 3.6, *no other optical-flow specific modifications were used.* No post-processing other than a median filter was needed.

3.7.3 Stereo

The main difference of the Stereo model from Optical Flow, is that it does not include the MRF-A terms, making the optimization easier. We tested our algorithm on the Middlebury Stereo 2014 Scharstein et al. (2014) and 2011 Baker et al. (2011) datasets

Method	Final pass					Clean pass				
	all	noc	occ	d0-10	s10-s40	all	noc	occ	d0-10	s10-s40
F3-MPLF (Ours)	6.272	3.092	32.207	5.042	**3.226**	4.770	2.062	26.863	3.459	**1.883**
FlowFields	**5.810**	**2.621**	31.799	**4.851**	3.739	3.748	**1.056**	25.700	2.784	2.110
FullFlow	5.895	2.838	**30.793**	4.905	3.373	3.601	1.296	**22.424**	2.944	2.055
DiscreteFlow	6.077	2.937	31.685	5.106	3.832	**3.567**	1.108	23.626	3.398	2.277
EpicFlow	6.285	3.060	32.564	5.205	3.727	4.115	1.360	26.595	3.660	2.117
TF+OFM	6.727	3.388	33.929	5.544	3.765	4.917	1.874	29.735	3.676	2.349
NNF-Local	7.249	2.973	42.088	4.896	4.183	5.386	1.397	37.896	**2.722**	2.245

Figure 3.3: Optical flow results on the test set of MPI Sintel dataset.

which are standard stereo matching datasets. The Stereo 2014 dataset is challenging because it includes 2864 $\times$ 1924 images with large movements. Fig.3.4 shows representative results from our algorithm. Our method does very well in finding the correct matchings for the earlier Stereo 2011 dataset with an overall AEE of just 0.07. On the hand the more recent 2014 dataset involves more iterations to solve because of the size of the datasets, but the results are comparable to those obtained from other methods.

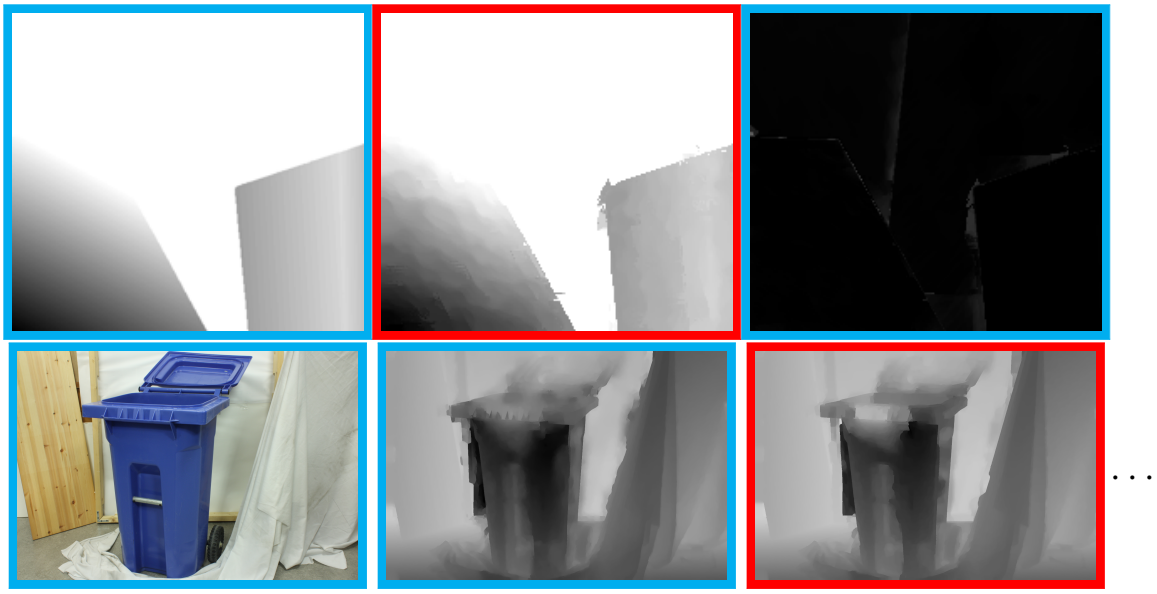


Figure 3.4: Stereo results: (From left to right) First row plots show the evaluation on Middlebury 2011 dataset. Our results match the ground truth closely; second row plots show the evaluation on Recycle pair from 2014 Middlebury dataset; the fifth image is the result after 10% of total epochs whereas the last image is the result after 50%. This implies that running more epochs progressively gives more accurate solutions. **Our results are marked in red.**

3.7.4 Affine Alignment

Recall that an affine warp is completely determined by 6 parameters. To test our algorithm, similar to Seitz and Baker (2009), we artificially warped an image with an affine transformation and then analyzed the error between the recovered $\hat{A}$ and true A^*. Results show that the quantity $||A^* - \hat{A}||_2^2$ was very small (within 10^{-3}) suggesting that the recovered warp from the Filter Flow model is close to the true warp. Our numerical scheme roughly takes $\approx$ 2 minutes to solve, compared to $\approx$ 3

hours mentioned in Seitz and Baker (2009). Overall, results presented here for 3 case studies show that our Filter Flow solver provides high quality solutions to various image transformation problems.

Running Time. Finally, we describe how our algorithm performs with respect to running time, which is arguably its most significant strength. For all experiments, we ran only 150 epochs of our algorithm: the sequential version of our algorithm takes $\approx$ 2 hours, whereas, our parallel implementation takes $\approx$ 10 minutes for a 640×480 image pair showing that the parallelized algorithm nearly gets a speedup proportional to #cores. It gives empirical evidence that 150 epochs are enough to get an approximate solution. Note that the corresponding times reported by Filter Flow Seitz and Baker (2009) is 9+ hours.

3.8 Conclusions

We proposed a reformulation of Filter Flow Seitz and Baker (2009), which makes it amenable to massively parallel asynchronous optimization schemes. Our reformulation is applicable to any problem expressible in the Filter Flow format. The generality does not involve any practical compromise — we show a range of experimental results demonstrating that a mostly *application-agnostic numerical solver can obtain results that are competitive* with specialized state of the art techniques. Compared to the original model Seitz and Baker (2009), the runtime reduces from several hours to a few minutes. Interestingly, the overall scheme is easy to implement and teach. A prototype version of our solver is only about 30 lines of Matlab code. The fully asynchronous solver will be made available as a fork of OpenCV. All the codes for this chapter are available in the Github page here.

Chapter 4

Coresets for Nonsmooth Problems

In this chapter, we generalize the CG Algorithm to handle nonsmooth objective functions in (Ravi et al., 2019a). We motivate our findings by drawing links to a recent line of research called *coresets*. Then, using a simple one dimensional example, we illustrate the main technical difficulty with this method and propose a natural fix to the algorithm. Our nonsmooth CG algorithm has an interesting property that when f is smooth, then the algorithm reduces to smooth CG algorithm, in Algorithm 4. Moreover, under the worst case scenario, our results match the best known rate of convergence using projected subgradient algorithm, in Algorithm 2.

4.1 Introduction

The expanding scope of numerical optimization applications (and the complexity of the associated data) has continued to raise the expectations of the efficiency profile of the underlying algorithms which drive the analyses modules in domains spanning genomics Banerjee et al. (2006), finance Pennanen (2012) and medicine Liu et al. (2009). For instance, the development of "low-order" polynomial time algorithms has long been a central focus of algorithmic research; the availability of a (near) linear time algorithm for a problem was considered a gold standard since any algorithm's runtime should, at a minimum, include the time it takes to evaluate the input data in its entirety. But as applications that generate extremely large data sets become more prevalent, we encounter many practical scenarios where even a procedure that takes only linear time to process the data may be considered impractical Sarlos (2006); Clarkson and Woodruff (2015). Within the last decade or so, there has been a lot of

effort to understand whether the standard notions of an efficient algorithm (as we saw in Chapter 2) are sufficient. In Chapter 3, we saw how algorithmic reformulations can be used for faster optimization. Along similar lines, efforts to study whether linear time algorithms are good enough for various turnkey applications has led to the study of so-called "sublinear" algorithms and a number of interesting results have emerged, see Clarkson et al. (2012); Rudelson and Vershynin (2007); Bachem et al. (2018); Baykal et al. (2018).

Sublinear computation is based upon the premise that a fast (albeit approximate) solution may, in some cases, be preferrable to the optimal solution obtained at a higher computational or financial cost. The strategy is a good fit in at least two different regimes: (a) operations on streaming data where an algorithm must run in real time (an inexact but prompt answer may suffice) Braverman and Chestnut (2014) or (b) where the appropriate type of data (for the task) is scarce or unavailable at sample sizes necessary like distribution testing Acharya et al. (2015); Daskalakis et al. (2018). In either case, an important feature of most sublinear algorithms is the use of an *approximated version* of the decision version of the original problem. Many of the technical results, for specific problems/tasks have focused on deriving so-called "property testing" schemes as a means to establish what an algorithm can be expected to accomplish without reading through all of the data. This line of work has led to many fast sublinear algorithms for numerical linear algebra problems including matrix multiplication Drineas et al. (2006), computing spectral norms and leading singular vectors of the data matrix Drineas et al. (2006) among others, and has been deployed in settings where the full data may have a large memory footprint. More recently, several authors have even shown how to extend this idea for solving convex optimization problems. In particular, the work by Clarkson (2008) which motivates our project obtains sublinear algorithms for a variety of problems where the constraints represent the unit probability simplex. Next, we briefly describe this framework and its relationship to *Coresets*, an idea often used in the design and analysis of algorithms in computational geometry, see Chan (2018).

4.1.1 Related Work

In an important result several years ago Clarkson (2008), the authors showed that sublinear algorithms can be designed for a variety of problems using the framework of the well known Frank-Wolfe (FW) algorithm Frank and Wolfe (1956). Recall that

FW algorithms can be used to solve smooth convex optimization problems when the feasible set is *compact*. Using this framework, Clarkson unified the analysis of various problems in machine learning and statistics including regression, boosting, and density mixture estimation via reformulations as optimization problems with a smooth convex objective function and where the unit simplex was the feasible set. This setup was then shown to provide immediate results to important theoretical and practical issues including sample complexity bounds and faster algorithms for Support Vector Machines and approximation bounds for boosting procedures. This result raises an interesting question: what properties of FW algorithms enable one to design fast algorithms with provable guarantees? Clarkson addressed this question by drawing a contrast between FW algorithms and a (widely used) alternative strategy, i.e., projected gradient type algorithms. Specifically, instead of a quadratic function, in FW algorithms, we optimize a linear function over the feasible set which often yields a better per iteration complexity for many problems (further, the iterates always remain feasible). It turns out that these properties are particularly useful from a optimization point of view since one can frequently obtain pipelines with significantly better *memory* complexity Garber and Meshi (2016); Gidel et al. (2017), as we discuss next.

An interesting consequence of the above work was a result showing a nice relationship of FW-type schemes to a concept predominantly used within computational geometry known as *coresets* Agarwal et al. (2005). Intuitively, a coreset (typically defined in the context of a problem statement) is a subset of the input data on which an algorithm with provable approximation guarantees for the problem at hand can be obtained which has a runtime/memory complexity that is, in some sense, independent of (or only loosely dependent on) the size of the dataset Huggins et al. (2016). A key practical consequence of the analysis in Clarkson et al. (2012) was that coresets for SVMs derived using the proposed procedure were tighter than all earlier results Tsang et al. (2005). For other machine learning problems which can be solved using convex optimization included in Clarkson et al. (2012), the analysis typically allows bounding the number of nonzero elements in the decision variables (independent of the total number of such variables). Clearly, these are very useful properties — but we see that the setup in Clarkson et al. (2012) requires that the objective function of the optimization problem be *differentiable*; this is somewhat restrictive for machine learning and computer vision applications, where various nonsmooth regularizers are ubiquitous and serve to impose structural or statistical requirements on the opti-

mal solution Bach et al. (2012a). The overarching goal of this chapter is to obtain an algorithm (very similar to the standard FW procedure) which makes no such assumption. Incidentally, we are also able to obtain coreset-type results for a much broader class of problems (that involve nonsmooth objective functions) — providing sublinear algorithms for several of these applications and sensible heuristics for others.

4.1.2 Overview of this work and related results

We first define the problem considered here and give an overview of some of the related ideas from a mathematical perspective. Let f be a convex and real-valued function (but *not* necessarily continuously differentiable) over a compact convex domain D. We study a specific class of finite dimensional optimization problems expressed as,

$$\min_{x \in D} f(x). \tag{4.1}$$

Note that we can assume f to be $L-$lipschitz for some $0 \leqslant L < \infty$ since D is compact. Our central result gives a new convergence bound for an algorithm derived from a scheme first outlined in White (1993) in the early 1990s but which seems to have been utilized only sparingly in the literature since. At a high level, our deterministic method generates a sequence of iterates x_k for $k = \{0, 1, 2, ...\}$ that provably converges in the primal–dual gap, which implies convergence in the objective function value as well since the problem is convex, by assumption. Specifically, if x_* is a solution to (4.1) we show that $f(x_k) \leqslant f(x_*) + O(1/\sqrt{k})$, thereby providing an *a priori* bound on the suboptimality of each iterate. We can restate this bound, for any choice of $\epsilon > 0$ and

$$f(x_k) \leqslant f(x_*) + \epsilon \text{ for any } k \geqslant K, \tag{4.2}$$

for some $K \in O(1/\epsilon^2)$.

Relevance. A wide range of problems can be expressed in the form shown in (4.1), and for some of these problems, bounds as in (4.2) will also yield a coreset result. As briefly described in the previous section, within a coreset-based algorithm, we can perform training on a very large numbers of examples while requiring computational resources that depend on the intrinsic hardness of the problem *regardless of the amount of data collected*. We note that while a subsampling heuristic may show this behavior for some datasets (and for some objective functions), in general, a coreset (if it exists) will typically yield a stronger approximation than ordinary subsampling.

For example, coreset results have been used for k-means and k-median clustering Har-Peled and Kushal (2005), subspace approximation Feldman et al. (2010), support vector machines and its variants Tsang et al. (2005); Nathan and Raghvendra (2014); Har-Peled et al. (2007), and robotics Feldman et al. (2012); Balcan et al. (2013). For several problems we discuss here, convergence bounds on the optimization model show that a solution x_K can be found with no more than K nonzero entries, and these nonzeros correspond to the choice of a coreset as mentioned above. Another consequence of such a result is that the coreset bound will be deterministic, giving a *tight* bound on the actual approximation error, as opposed to a bound on the *expected* approximation error. For some problems, our results also translate well to practice, yielding fast implementations discussed in the experimental section.

Other generic smoothing techniques. We will go in to more detail on the some of the standard techniques to handle nonsmooth objective function f, previously described in Section 2.1.6. A small set of results pertaining to nonsmooth Frank Wolfe algorithms (and our proposals) have been reported in the literature in the last few years. At the high level, these approaches correspond to three different flavors:

1. **Noisy gradients:** Jaggi (2013) considers the case where the FW method uses an inexact gradient such that the linear subproblems provide a solution to the exact problem within a bounded error but assume that the objective function is *smooth*.

2. **Dualizing:** Jaggi (2011) presented a FW dual for nonsmooth functions employing the *subgradients* of the objective. This can be used to generate a "certificate" of optimality of an approximate solution via the primal-dual gap but the algorithms do not deal with the nonsmoothness directly. In particular, for the applications studied in Jaggi (2011), the nonsmooth parts in the objective function are dualized instead and treated as constraints. This approach becomes inefficient since the subproblems are complicated for many other important applications that are considered here.

3. **Smoothing:** Hazan and Kale (2012) presents an algorithm for minimization of nonsmooth functions with bounded regret that applies the FW algorithm to a randomly smoothed objective, instead providing probabilistic bounds. This approach closely addresses the setting that we consider, however, it has a practical bottleneck. Observe that (at each iteration) one has to make many

queries to the zeroth order oracle of the objective function in order to compute the gradient of the smoothed (approximate) function which is often expensive when the function depends on the number of data points (may be quite large in many problems). We do observe this behavior in our experimental evaluations in Section 7.5.

We will point out other related works in the relevant sections below. Overall, we see that much of the existing literature has approached this problem either via an implicit *randomized smoothing* Hazan and Kale (2012) or by using proximal functions Argyriou et al. (2014); Pierucci et al. (2014) to yield suitable gradients and provide convergence results for the *smoothed* objective. Contemporary to our work, Cheung and Li (2017) developed a Nonsmooth Frank-Wolfe algorithm for minimizing ℓ_1-regularized smooth convex functions over the trace norm ball. The update directions in Cheung and Li (2017) are obtained by considering all affine functions over the ℓ_∞ ball centered around the current iterate with the radius chosen appropriately to ensure convergence. Our approach can be seen as a generalization of Cheung and Li (2017) in that we show that if we choose a neighborhood that is problem specific (instead of ℓ_∞ ball), we can show both convergence and coreset results. To that end, we will discuss a general class of optimization problems for which our algorithm can be viable in Section 4.4.6. Interestingly, subproblems that arise in our algorithm can also be solved efficiently in theory and practice for all the applications discussed in Cheung and Li (2017) as we will see shortly in Section 4.4.

Our Results. In Section 7.2, we discuss why simply using a subgradient is insufficient to prove convergence. Then, we introduce the basic concepts that are used in our algorithm and analysis. The starting point of our development is a specific algorithm mentioned in White (1993), which is of the same general form as Algorithm 6 of this chapter in Section 4.3; White (1993) includes a result equivalent to the "Approximate Weak Duality" that we describe later. In Section 4.3, we derive an *a priori* convergence result for a nonsmooth generalization of the FW method, which also yields a *deterministic* bound on the approximation error and the size of the constructed coreset. We use a much more general construction for the approximate subdifferential $T(x, \epsilon)$ and show how this can yield novel *coreset* results analogous to those shown for the FW method in the smooth case in Clarkson (2008). Using these results, we analyze many important problems in statistics and machine learning in Section 4.4. While the algorithm we present may not be a silver bullet for arbitrary

nonsmooth problems (which may have *specialized* algorithms), in several settings, the results do have practical import, these are discussed in Section 7.5. For instance, we show an example where our algorithm enables solving very large problem instances on a single desktop where other reported approaches have deployed distributed optimization schemes on a cluster. On the theoretical side, a useful result is that our scheme can produce coresets with hard approximation bounds.

4.2 Preliminary Concepts

To introduce our algorithm and the corresponding convergence analysis, we start with some basic definitions first. Recall that a fundamental tool for optimization of nonsmooth functions is the *subdifferential*. It is well known that it is possible to design algorithms with $O(1/\epsilon^2)$ convergence rate for nonsmooth problems, see Nesterov (1998); Lan et al. (2012). Unfortunately, utilizing just the subdifferential, turns out to be insufficient to produce the necessary convergence bounds for our analysis. For instance, in a recent paper Nesterov (2015), the authors constructively showed that simply replacing the gradient by a subgradient in a FW algorithm is incapable of ensuring convergence — a simple two dimensional example demonstrated that the FW algorithm does *not* converge to the optimal solution for *any* number of iterations. To our knowledge, there are no clear strategies to fix this limitation. As mentioned above, one often uses smoothing techniques as a workaround, where an auxiliary function is constructed which is optimized instead Nesterov (2005); Pierucci et al. (2014). While this approach has had significant practical impact, it involves the selection of a *proximal* function. But there are no general recipes to choose the proximal function for a given model and is often designed based on the specific problem at hand.

As we saw in Chapters 2 and 3, the convergence analysis of FW type methods rely on the boundedness of an important quantity called the "Curvature" Constant,

$$C_f^{\text{exact}} := \sup_{\substack{x,\, s \in D \\ \alpha \in [0,1] \\ y = x + \alpha(s-x)}} \left(\min_{d \in \partial f(x)} \frac{1}{\alpha^2} \left(f(y) - f(x) - \langle y - x, d \rangle \right) \right).$$

It describes how well the first-order information from the approximate subdifferential *globally* describes the function (when f is smooth, d is simply the gradient of f). It is

this quantity that becomes unbounded even for simple nonsmooth functions making the subsequent analysis difficult. To make this point clear, we now show a simple example of a nonsmooth function for which this quantity is not bounded, though in this 'easy' case we can prove convergence with more direct means.

Example 1. *Let* $f(x) = |x|$ *over* $D = [-1, 1]$. *For any* $x \in (0, \frac{1}{2})$, *let* $s = -1 + x$ *and* $\alpha = 2x$. *Accordingly,*

$$y = x + \alpha(s - x) = x + 2x(-1 + x - x) = -x.$$

By definition,

$$C_f^{\text{exact}} \geqslant \min_{d \in \partial f(x)} \frac{1}{\alpha^2} \left(f(y) - f(x) - \langle y - x, d \rangle \right) = \min_{d \in \partial f(x)} \frac{1}{4x^2} \left(f(-x) - f(x) + \langle 2x, d \rangle \right).$$

Because f *is differentiable at* x, $\partial f(x) = \{1\}$,

$$C_f^{\text{exact}} \geqslant \frac{1}{4x^2}(|-x| - |x| + 2x) = \frac{1}{2x}.$$

Hence we have that,$\lim_{x \to 0^+} \frac{1}{2x} = +\infty$, *showing that we cannot obtain an upper bound* C_f^{exact} *for all* $x \in (0, \frac{1}{2})$. *This example shows that linear approximations based on ordinary subgradients can be arbitrarily poor in the neighborhood of nondifferentiable points.*

Basic idea. Intuitively, the subdifferential is a discontinuous multifunction Robinson (2012) and provides an incomplete description of the curvature C_f of the function in the neighborhood of the nonsmooth points. After making a few technical adjustments, it turns out that we can work around this issue by making use of *approximate* subdifferentials. We deal with two constructions that yield approximate subdifferentials. Using these definitions, we will give the formal statement of our convergence theorem in the next section. The first is the ϵ-*subdifferential* definition given in Hiriart-Urruty and Lemaréchal (1993),

Definition 4.1 (ϵ-subdifferential)**.** *The* ϵ-*subdifferential* $\partial_\epsilon f(x)$ *of* f *at* $x \in$ *dom* f *is given by,*

$$\partial_\epsilon f(x) := \{d \mid f(y) \geqslant f(x) + \langle d, y - x \rangle - \epsilon \ \forall\, y\}.$$

Notice that the exact subdifferential is recovered when ϵ is zero, $\partial_0 f(x) = \partial f(x)$.

While the ϵ-subdifferential indeed provides the theoretical properties for our convergence bounds, in practical cases, it requires us to work with carefully chosen problem-specific subsets of $\partial_\epsilon f(x)$. A successive construction which we present later produces approximate subdifferentials that have nicer computational properties while preserving the convergence bounds. In this approximation, we must take the subdifferentials over a neighborhood; this idea is summarized next.

Let N be an arbitrary mapping from x to neighborhoods around x such that $N(x, \epsilon) \to \{x\}$ as $\epsilon \to 0$. Formally, we assume that $x \in N(x, \epsilon)$ and there exists some constant R such that $N(x, \epsilon) \subseteq \mathcal{B}(x, R\epsilon)$ for all $\epsilon > 0$. The notation $\mathcal{B}(x, r)$ is the open ball around x of radius r. We may assume w.l.o.g. that $R = 1$ wherever appropriate.

Definition 4.2. *The approximate subdifferential* T *of* f *at* x *is defined as,*

$$T(x, \epsilon) := \begin{cases} \{\nabla f(x)\} & \text{if } f \text{ is differentiable on } N(x, \epsilon), \\ \bigcup_{u \in N(x,\epsilon)} \partial f(u) & \text{otherwise.} \end{cases}$$

Here T provides a set of approximate subgradients of f at x. Indeed, if f is L-Lipschitz, we have $T\left(x, \frac{\epsilon}{2L}\right) \subseteq \bigcup_{u \in \mathcal{B}\left(x, \frac{\epsilon}{2L}\right)} \partial f(u) \subseteq \partial_\epsilon f(x)$ by Theorem 8.4.4 of Robinson (2012). It turns out that with these constructions, we will shortly obtain a *deterministic* non-smooth FW algorithm with accompanying *convergence* results.

4.3 Convergence Results

Using the foregoing concepts, we can adapt the procedure in White (1993) with a few minor modifications as shown in Algorithm 6.

Algorithm 6 Conditional Subgradient (based on ideas introduced in White (1993))

Pick an arbitrary starting point $x_0 \in D$, where D is the compact convex feasible set.
for $k = 0, 1, 2, \ldots$ **do**
Let $\alpha_k := \frac{2}{k+2}$ and $\epsilon_k := \sqrt{\alpha_k}$

$$s \in \arg\min_{z \in D} \max_{d \in T(x, \epsilon_k)} \langle z - x, d \rangle \tag{4.3}$$

Update $x_{k+1} := x_k + \alpha_k(s - x_k)$
end for

While both the Algorithm 6 and the one proposed in White (1993) are similar in spirit, White (1993) requires us to do *exact* line search which limits the applicability in our experimental settings. Moreover, the convergence result in White (1993) is asymptotic (as $k \to \infty$) whereas we provide explicit convergence rate both in primal and in duality gap. Our analysis, arguably, gives a better understanding of the algorithm using the affine invariant curvature concept.

To simplify the presentation, we will assume that the subproblems (4.3) are efficiently solvable both in theory and in practice. In general, this is true when $T(x, \epsilon)$ is a polyhedron since any point $d \in T(x, \epsilon)$ can be written as a convex combination of extreme points and extreme rays. While it might seem like this severely limits the applicability of the proposed approach (to a point where it is hard to imagine when it can be applied unless if the subdifferentials are polytopes), we identify a general class of problems in Section 4.4.6 for which the subproblems can be solved efficiently in theory even when the data set size is large. We also show shortly that for various statistical machine learning problems, we can solve (4.3) far more efficiently even when $T(x, \epsilon)$ is not a polyhedron making the overall algorithm attractive in any case. With this assumption, we may generalize the curvature constant which plays a key role in our convergence analysis,

Definition 4.3 ($C_f(\epsilon)$)**.** *The Approximate Curvature Constant* $C_f(\epsilon)$ *of* f *is defined as,*

$$C_f(\epsilon) := \sup_{\substack{x,s \in D \\ \alpha \in [0,1] \\ y = x + \alpha(s-x)}} \min_{d \in T(x,\epsilon)} \frac{1}{\alpha^2} \left(f(y) - f(x) - \langle y - x, d \rangle \right). \tag{4.4}$$

Importantly, we use approximate subgradients instead of a gradient as in Clarkson (2008), where the value of C_f depends on ϵ. We can choose an ϵ varying with iterations in such a way as to guarantee convergence although in practical problems, we see that $C_f \in O(1/\epsilon)$. In any case, note that the complexity of the algorithm does *not* depend on the input data, that is, we can choose ϵ to be very small such that optimizing linear functions over $T(x, \epsilon)$ is tractable in practice. Concretely, input data in our settings correspond to either training examples (or features) depending on the application. In particular, when the decision variables are defined over examples, and the subdifferential can be computed with *only* the nonzero coordinates of the decision variables, our algorithm ends up being *sublinear* in the number of training examples (or features) depending on the application. We will see this in detail in the

later sections for specific problems that are also empirically verified in Section 7.5. Our central convergence theorem, stated in terms of this constant, appears next.

Theorem 4.4. *Suppose* f *is* L*-Lipschitz and* $C_f(\epsilon) \leqslant \frac{D_f}{\epsilon}$ *for some constant* D_f *for any* $\epsilon \leqslant 1$*. Then Alg. 6 generates a sequence of iterates* x_k*, such that for the* k^{th} *iteration:*

$$f(x_k) - f(x_*) \leqslant \frac{2^{5/2}L + 2^{3/2}D_f}{\sqrt{k+2}}, \tag{4.5}$$

where x_* *is the optimal solution to the problem in (4.1).*

Discussion. As the iteration count k increases, the denominator in this upper bound increases. Since the numerator consists of constants depending only on the definition of f and D, this will approach 0 as $k \to \infty$. It is important to notice that while this theorem provides a general convergence bound (which is useful), it does *not* alone prove that Algorithm 6 will produce a *coreset* and consequently, will be sublinear. Indeed, it is a known result for *smooth functions*. Shortly, in example cases covered in Section 4.4, we show *the other key component* of the proposed algorithm: a bound on the number of nonzeros for any solution s of the subproblems (4.3). The key results here will depend on D and f, and we will seek to show that the number of nonzero entries in s will be $O(1)$ w.r.t. the overall size of the problem. This in turn guarantees that x_k will have no more than $O(k)$ nonzeros. Combined with the convergence bound above, this in turn will demonstrate that a sparse ϵ-approximate solution can be found with $O(1/\epsilon^2)$ nonzeros which does *not* depend on the problem size (e.g., number of samples/examples or dimensions). When x is a vector of the *examples* in a machine learning problem, the union of nonzeros for each s found by Algorithm 6 will constitute a *coreset*. Theorem (4.4) can be used as a *general framework* that can be extended to show a coreset result for any nonsmooth problem for which a sparsity bound on s can in turn be shown, yielding a valuable tool for sublinear algorithm design.

Ingredients. The proof of Theorem 4.4 relies on a bound in the improvement in the objective at each iteration in terms of $C_f(\epsilon)$ (in (4.4)) and the duality gap between the primal objective at x and a nonsmooth modification of the Wolfe dual. We define this dual ω of f as:

$$\omega(x, \epsilon) := \min_{z \in D} \max_{d \in \partial_\epsilon f(x)} f(x) + \langle z - x, d \rangle. \tag{4.6}$$

This dual gives us a property we call *approximate weak duality*. Up to ϵ, this dual is at all points less than the minimum value of the primal objective at all points.

Lemma 4.5 (Approx. Weak Duality)**.** $\omega(x, \epsilon) \leqslant f(y) + \epsilon$ *for all* $x, y \in D$,

Proof. Take any $x, y \in D$. Due to the minimization in (4.6),

$$\omega(x, \epsilon) \leqslant \max_{d \in \partial_\epsilon f(x)} f(x) + \langle y - x, d \rangle.$$

By the definition of the ϵ-subdifferential, for any $d \in \partial_\epsilon f(x)$, including the one chosen by the maximization,

$$f(x) + \langle y - x, d \rangle \leqslant f(y) + \epsilon,$$

which gives us the desired result. □

Denote the primal-dual gap at x by

$$g(x, \epsilon) := f(x) - \omega(x, \epsilon) = \max_{y \in D} \min_{d \in \partial_\epsilon f(x)} \langle x - y, d \rangle. \tag{4.7}$$

Then, by Lemma 4.5,

$$g(x, \epsilon) \geqslant f(x) - f(x_*) - \epsilon \geqslant -\epsilon.$$

The combination of the primal-dual gap and the curvature constant C_f is used to show per iteration improvement on the objective. This is then used to show the a priori convergence result in Theorem 4.4. We see that at each step of Algorithm 6, the objective at x_{k+1} improves upon the objective at the previous iterate x_k by a term proportional to the primal-dual gap, up to a "curvature error" term with C_f. This is stated in the following result:

Theorem 4.6. *Fix* $\epsilon' = 2L\epsilon_k$. *For any step* $x_{k+1} := x_k + \alpha(s - x_k)$, *with arbitrary step size* $\alpha \in [0, 1]$ *and* $s \in D$ *chosen as in* (4.3), *it holds that:*

$$f(x_{k+1}) \leqslant f(x_k) - \alpha g(x_k, \epsilon') + \alpha^2 C_f(\epsilon_k). \tag{4.8}$$

Proof. Write $x := x_k, y := x_{k+1} = x + \alpha(s - x)$, and $\epsilon := \epsilon_k$. From the definition of C_f,

$$\begin{aligned} f(y) &\leqslant f(x) + \left(\max_{d \in T(x,\epsilon)} \langle y - x, d \rangle \right) + \alpha^2 C_f(\epsilon) \\ &= f(x) + \alpha \left(\max_{d \in T(x,\epsilon)} \langle s - x, d \rangle \right) + \alpha^2 C_f(\epsilon). \end{aligned}$$

By choice of s in (4.3),

$$f(y) \leqslant f(x) + \alpha \left(\min_{y \in D} \max_{d \in T(x,\epsilon)} \langle y - x, d \rangle \right) + \alpha^2 C_f(\epsilon).$$

Now using the fact that $T\left(x, \frac{\epsilon}{2L}\right) \subseteq \partial_\epsilon f(x)$,

$$\begin{aligned} f(y) &\leqslant f(x) + \alpha \left(\min_{y \in D} \max_{d \in \partial_{\epsilon'} f(x)} \langle y - x, d \rangle \right) + \alpha^2 C_f(\epsilon) \\ &= f(x) - \alpha \left(\max_{y \in D} \min_{d \in \partial_{\epsilon'} f(x)} \langle x - y, d \rangle \right) + \alpha^2 C_f(\epsilon). \end{aligned}$$

By definition in (4.7), we have that $f(y) \leqslant f(x) - \alpha g(x, \epsilon') + \alpha^2 C_f(\epsilon)$ as desired. □

Now we are ready to prove Theorem 4.4.

Proof of Theorem 4.4. Finally, we prove the main theorem by inductively applying the per step improvement in (4.8) to find the *a priori* bound in terms of k.

Let $h(x) = f(x) - f(x_*)$. Using Theorem 4.6, we first restate the stepwise improvement in the objective as follows:

$$h(x_{k+1}) \leqslant h(x_k) - \alpha_k g(x_k, 2L\epsilon_k) + \alpha_k^2 C_f(\epsilon_k).$$

By Lemma 4.5,

$$\begin{aligned} h(x_{k+1}) &\leqslant h(x_k) - \alpha_k (h(x_k) - 2L\epsilon_k) + \alpha_k^2 C_f(\epsilon_k) \\ &\leqslant (1 - \alpha_k) h(x_k) + 2L\alpha_k \epsilon_k + \frac{\alpha_k^2}{\epsilon_k} D_f. \end{aligned} \tag{$*$}$$

Now, if we substitute the constant $E := 2L + D_f$, and the choice $\epsilon_k = \sqrt{\alpha_k}$ as in the algorithm,

$$h(x_{k+1}) \leqslant (1 - \alpha_k)h(x_k) + \alpha_k^{3/2}E.$$

(We will later see in the experiments how this choice of ϵ_k makes the algorithm practically useful.) Using this notation to rewrite the theorem, we seek to show $h(x_k) \leqslant \frac{2^{3/2}E}{\sqrt{k+2}} = 2E\sqrt{\alpha_k}$. We prove this by induction. The base case comes from applying (4.8) with $k = 0$ and $\alpha_k = 1$ to get $h(x_1) \leqslant E \leqslant \frac{2^{3/2}E}{\sqrt{3}}$. For $k \geqslant 1$, making the induction step,

$$\begin{aligned} h(x_{k+1}) &\leqslant (1 - \alpha_k)\left(2E\sqrt{\alpha_k}\right) + \alpha_k^{3/2}E \\ &= \left(1 - \frac{2}{k+2}\right)\frac{2^{3/2}E}{\sqrt{k+2}} + \frac{2^{3/2}}{(k+2)^{3/2}}E \\ &= 2^{3/2}E\sqrt{k+2}\left(\frac{1}{k+2} - \frac{1}{(k+2)^2}\right). \end{aligned}$$

Now we make an elementary calculation,

$$\begin{aligned} \left(\frac{1}{k+2} - \frac{1}{(k+2)^2}\right) &= \frac{1}{k+2}\frac{k+2-1}{k+2} \\ &\leqslant \frac{1}{k+2}\frac{k+2}{k+3} = \frac{1}{k+3}. \end{aligned} \tag{4.10}$$

Using (4.10), we can get the desired result as,

$$h(x_{k+1}) \leqslant 2^{3/2}E\frac{\sqrt{k+2}}{k+3} \leqslant 2^{3/2}E\frac{\sqrt{k+3}}{k+3} = \frac{2^{3/2}E}{\sqrt{k+3}}.$$

□

Discussion. A key intuition about the algorithm can be gained from the proof step in (∗). This inequality implies that the improvement in the objective for each step has a bounded "error" with two parts. First is an error linear in α_k that comes from our use of an approximate subgradient rather than an exact subgradient. This error is proportional to ϵ_k. Second, we have a "curvature" error that arises from the nonlinearity (specifically nonsmoothness), of our objective function and is instead *inversely* proportional to ϵ_k. By choosing a decaying ϵ_k such that $\frac{1}{\epsilon_k} \in \Theta\left(\frac{1}{\sqrt{\alpha_k}}\right)$ we

send both error terms to 0 as $k \to \infty$.

4.4 Approximable Nonsmooth Problems

In this section, we demonstrate how the foregoing ideas can be applied and made more specific, on a case by case basis, for various problems arising in statistical machine learning. Since many of the steps and quantities in the algorithm and theorems described in the previous section depend on details of the individual problem, we use these examples to show the potential value of our results. In this section, each problem we present yields a choice of objective function f and feasible region D. For these choices, we then describe the approximate subdifferentials and the resulting subproblems to determine the step direction. We show that the antecedents of Theorem 4.4 are satisfied for a number of machine learning problems and provide $O(1/\epsilon)$ bounds for $C_f(\epsilon)$. A problem-specific bound on C_f is the final component of establishing an iteration bound for that problem. It guarantees that our scheme for selecting ϵ_k will provide convergence, and that we have finite values for the numerator in (4.5). Therefore, the quotient will approach 0 in the limit.

Overview of problems. We first analyze the supremum of linear functions with the goal of getting an expression that bounds C_f from above. With this in hand, we will present a formulation of ℓ_1 SVM that is suitable for our analysis. In particular, we show C_f bounds using the result from piecewise linear functions and then proceed to discuss how the subproblems (4.3) can be efficiently solved. We discuss the tractability and provide tight coreset bounds for this problem. Later, we analyze the multiway graph cuts model by adapting the techniques used in the ℓ_1 SVM problem and finally, we also describe how the analysis extends to the 1-median problem, Sparse PCA and Balanced development.

4.4.1 Supremum of Linear Functions

We can define a piecewise linear convex function on $\mathbb{R}^n$ by

$$f(x) := \max_{i=1,\dots,p} a_i^T x + b_i. \tag{4.11}$$

If we let $A \in \mathbb{R}^{p\times n}$ be the matrix such that row i is equal to a_i, and $b \in \mathbb{R}^p$ be the vector that has elements b_i. We know that f is Lipschitz-smooth with constant equal

to the operator norm $\|A\|$ and $f(x) \equiv \hat{f}(Ax + b)$ for

$$\hat{f}(\hat{x}) = \max_{i=1,\dots p} \hat{x}_i \implies f(x) := \hat{f}(Ax + b). \tag{4.12}$$

The subdifferentials of $\hat{f}$ are given by

$$\partial \hat{f}(\hat{x}) = \left\{ e_i \;\middle|\; \left(\max_{j=1,\dots,p} \hat{x}_j \right) = \hat{x}_i \right\} \text{where } e_i \text{ is the } i\text{—th basis vector.} \tag{4.13}$$

For the purposes of calculating approximate subdifferentials $\hat{T}$ and T of $\hat{f}$ and f respectively (defined in section 7.2), we use the neighborhoods:

$$\hat{N}(\hat{x}, \epsilon) = \{\hat{u} \mid \|\hat{u} - \hat{x}\|_\infty \leqslant \epsilon\}, \quad N(x, \epsilon) = \{u \mid \|Au - Ax\|_\infty \leqslant \epsilon \text{ and } u - x \in \operatorname{im}\left(A^T\right)\}, \tag{4.14}$$

where by our standard notations in Section 1.4, A^T is the transpose of A and $\operatorname{im}\left(A^T\right)$ is the row space of A.

Theorem 4.7. *Under the setting of* (4.11),(4.12), (4.14), *define*

$$V(\hat{x}, \epsilon) = \left\{ e_i \;\middle|\; \hat{x}_i \geqslant \left(\max_j \hat{x}_j \right) - 2\epsilon \right\},$$

and let $\mathrm{Co}S$ *denote the convex hull of the set* S. *Then, the approximate subdifferential is given by* $\hat{T}(\hat{x}, \epsilon) = \mathrm{Co}V(\hat{x}, \epsilon)$.

Proof. Note that both sides will be a singleton set equal to $\{\nabla \hat{f}(\hat{x})\}$ iff $\hat{f}$ is differentiable on $\hat{N}(\hat{x}, \epsilon)$. We therefore prove only the case that $\hat{f}$ is nondifferentiable somewhere in the neighborhood.

Forward Direction. $\hat{T}(\hat{x}, \epsilon) \subseteq \mathrm{Co}V(\hat{x}, \epsilon)$.

Take any $\hat{u} \in \hat{N}(\hat{x}, \epsilon)$. We have,

$$\partial \hat{f}(\hat{u}) = \mathrm{Co}\left\{ e_i \;\middle|\; \hat{u}_i = \left(\max_j \hat{u}_j \right) \right\}. \tag{4.15}$$

Now take any i such that $e_i \in \operatorname{vert}(\partial \hat{f}(\hat{u}))$ where $\operatorname{vert}(S)$ denotes the vertices of a polyhedral set S as in Section 1.4. If S is a finite set of linearly independent points, then $\operatorname{vert}(\mathrm{Co}S) = S$ (which in this case is the set of basis vectors in (4.15)). Because

$\|\hat{u} - \hat{x}\|_\infty \leqslant \epsilon$,

$$\hat{x}_i \geqslant \hat{u}_i - \epsilon = \left(\max_j \hat{u}_j\right) - \epsilon \geqslant \left(\max_j \hat{x}_j\right) - 2\epsilon.$$

Therefore,

$$\text{vert}(\partial \hat{f}(\hat{u})) \subseteq V(\hat{x}, \epsilon) \implies \partial \hat{f}(\hat{u}) \subseteq \text{CoV}(\hat{x}, \epsilon). \tag{4.16}$$

Since (4.16) is true for all $u \in \hat{N}(\hat{x}, \epsilon)$, we see that $\hat{T}(\hat{x}, \epsilon) \subseteq \text{CoV}(\hat{x}, \epsilon)$.
Reverse Direction. $\hat{T}(\hat{x}, \epsilon) \supseteq \text{CoV}(\hat{x}, \epsilon)$
Let $\hat{u}$ be the vector defined elementwise by,

$$\hat{u}_i = \begin{cases} (\max_j \hat{x}_j) - \epsilon & \text{if } \hat{x}_i \geqslant (\max_j \hat{x}_j) - 2\epsilon, \\ \hat{x}_i & \text{otherwise.} \end{cases}$$

If we take some i such that $\hat{x}_i \geqslant (\max_j \hat{x}_j) - 2\epsilon$, then,

$$\begin{aligned} \hat{u}_i - \hat{x}_i &= \left(\max_j \hat{x}_j\right) - \epsilon - \hat{x}_i \\ &\leqslant \left(\max_j \hat{x}_j\right) - \epsilon - \left(\left(\max_j \hat{x}_j\right) - 2\epsilon\right) = \epsilon. \end{aligned}$$

Similarly, it also holds that,

$$\hat{x}_i - \hat{u}_i \leqslant \left(\max_j \hat{x}_j\right) - \hat{u}_i = \left(\max_j \hat{x}_j\right) - \left(\left(\max_j \hat{x}_j\right) - \epsilon\right) = \epsilon.$$

And $\hat{u}_i = \hat{x}_i$ for the "otherwise" case. Therefore $\|\hat{u}_i - \hat{x}_i\|_\infty \leqslant \epsilon$ and $\hat{u}_i \in \hat{N}(\hat{x}, \epsilon)$. Furthermore, note that clearly $\hat{f}(\hat{u}) = \max_j \hat{x}_j - \epsilon$, and for any i such that

$$\hat{x}_i < \left(\max_j \hat{x}_j\right) - 2\epsilon,$$

we have that, for all $\epsilon > 0$,

$$\hat{x}_i < \left(\max_j \hat{x}_j\right) - \epsilon = \hat{f}(\hat{u}).$$

Accordingly,

$$\text{vert}(\partial\hat{f}(\hat{u})) = \hat{V}(\hat{x},\epsilon) \implies \hat{T}(\hat{x},\epsilon) \supseteq \partial\hat{f}(\hat{u}) = \text{Co}\hat{V}(\hat{x},\epsilon).$$

□

Lemma 4.8. *Under the setting of* (4.11),(4.12), (4.14), $T(x,\epsilon) = A^T\hat{T}(Ax+b,\epsilon)$.

Proof. This follows from the fact that $\partial f(u) = A^T\partial\hat{f}(Au+b)$, $AN(x,\epsilon)+b = \hat{N}(Ax+b,\epsilon)$. □

Corollary 4.9. *Under the setting of* (4.11),(4.12), (4.14), $T(x,\epsilon) = A^T\text{Co}V(Ax+b,\epsilon) = \text{Co}A^TV(Ax+b,\epsilon)$.

Proof. The first equality is simply the combination of Lemma 4.8 and Theorem 4.7. The second is clear from the definition of convex hull. □

So the approximate subdifferentials $\hat{T}$ and T, of the functions $\hat{f}$ and f respectively are related by,

$$\hat{T}(\hat{x},\epsilon) = \text{Co}V(\hat{x},\epsilon) \quad \text{and} \quad T(x,\epsilon) = A^T\text{Co}V(Ax+b,\epsilon), \tag{4.17}$$

for an appropriate choice of neighborhoods.

Lemma 4.10. *Under the setting of* (4.11),(4.12), (4.14), *if* $\hat{y} \in \hat{N}(\hat{x},\epsilon)$, *then* $\partial\hat{f}(\hat{y}) \subseteq \hat{T}(\hat{x},\epsilon)$.

Proof. If $\hat{f}$ is nondifferentiable anywhere on $\hat{N}(\hat{x},\epsilon)$, the inclusion follows from the definition of $\hat{T}$. So, we will consider the case when the neighborhood contains no non-differentiable points. Let $i' := \arg\max_i \hat{x}_i$. If $\hat{f}$ is instead differentiable everywhere on the neighborhood around $\hat{x}$, this means $\hat{u}_i < \hat{u}_{i'}$ for all $u \in \hat{N}(\hat{x},\epsilon)$ for any $i \neq i'$. Therefore, $\nabla\hat{f}(\hat{x}) = \nabla\hat{f}(\hat{y}) = \{e_{i'}\}$, and the lemma is true as an equality. □

Theorem 4.11. *Under the setting of* (4.11),(4.12), (4.14), *let* $\hat{x},\hat{s} \in \mathbb{R}^p$ *and* $\alpha \in [0,1]$. *Let* $\hat{y} := \hat{x} + \alpha(\hat{s}-\hat{x})$. *Then,*

$$\min_{\hat{d}\in\hat{T}(x,\epsilon)} \frac{1}{\alpha^2}\left(\hat{f}(\hat{y}) - \hat{f}(\hat{x}) - \langle\hat{y}-\hat{x},\hat{d}\rangle\right) \leqslant \frac{2}{\epsilon}\|\hat{s}-\hat{x}\|_\infty^2. \tag{4.18}$$

Proof. It is sufficient to look at the following two cases,

Case 1. $\hat{y} \in \hat{N}(\hat{x}, \epsilon)$.

Let $i' = \arg\max_i \hat{y}_i$. Then $e_{i'} \in \partial\hat{f}(\hat{y}) \subseteq \hat{T}(x, \epsilon)$ from (4.13) and Lemma 4.10. Then the left hand side of (4.18) is bounded above by

$$\hat{f}(\hat{y}) - \hat{f}(\hat{x}) - \langle \hat{y} - \hat{x}, e_{i'} \rangle = \hat{y}_{i'} - \left(\max_i \hat{x}_i\right) - (\hat{y}_{i'} - \hat{x}_{i'}) = \hat{x}_{i'} - \left(\max_i \hat{x}_i\right) \leqslant 0.$$

Case 2. $\hat{y} \notin \hat{N}(\hat{x}, \epsilon)$.

First note that in this case we have a lower bound on α:

$$\alpha \geqslant \frac{\epsilon}{\|\hat{s} - \hat{x}\|_\infty}. \tag{4.19}$$

Then we use the fact that $\|\hat{d}\|_1 \leqslant 1$ for any $\hat{d} \in \partial\hat{f}(\hat{u})$ for any $\hat{u} \in \mathbb{R}^p$ to bound the left hand side of (4.18) by,

$$\begin{aligned}\hat{f}(\hat{y}) - \hat{f}(\hat{x}) + \|\hat{y} - \hat{x}\|_\infty &= \left(\max_i\ \hat{x}_i + \alpha(\hat{s}_i - \hat{x}_i)\right) - \left(\max_i \hat{x}_i\right) + \alpha\|\hat{s} - \hat{x}\|_\infty \\ &\leqslant \left(\max_i \hat{x}_i\right) + \alpha\left(\max_i \hat{s}_i - \hat{x}_i\right) - \left(\max_i \hat{x}_i\right) + \alpha\|\hat{s} - \hat{x}\|_\infty \leqslant 2\alpha\|\hat{s} - \hat{x}\|_\infty.\end{aligned} \tag{4.20}$$

The bound in the lemma then follows from (4.19) and (4.20). □

Remark 4.12. *Note that the lower bound of α in (4.19) is only for analyzing the curvature constant $C_f(\epsilon)$, and it has no consequence for implementation. Essentially, $C_f(\epsilon)$ is a quantity that depends only on f, D and does not depend on α i.e., (4.18) holds for any $\alpha \in [0, 1]$. The convergence result follows from Theorem 4.4 as long as $C_f(\epsilon) < \infty$.*

Let us denote the diameter of a set S with respect to the q-norm by $\mathrm{diam}_q(S) = \max_{x,y\in S} \|x - y\|_q$. Then, we can further bound (4.18) by the diameter of the bounded feasible set.

Corollary 4.13. *Given the problem of minimizing a piecewise linear function $f(x) = \max_i(Ax + b)_i$ over a bounded set D, the corresponding C_f is bounded above by $C_f(\epsilon) \leqslant \frac{2}{\epsilon}(\mathrm{diam}_\infty(A\,D))^2$.*

Proof. This comes from the definition,

$$\operatorname{diam}_p(S) = \max_{x,y \in S} \|x - y\|_p,$$

so that $\|\hat{s} - \hat{x}\|_\infty \leqslant \operatorname{diam}_\infty(A\ D)$, for all $\hat{s} = As + b, \hat{x} = Ax + b$ with $x, s \in D$. □

4.4.1.1 ℓ_1-norm-regularized SVM

The first machine learning problem we analyze is the ℓ_1-norm-regularized SVM. Since the ℓ_1 norm is sparsity-inducing, the regularization $\|w\|_1$ gives an optimization problem that finds a separating hyperplane in only a small subset of the features. This is expected to perform well in problems where there is a large number of redundant or noisy features included with a few informative features.

Suppose we are given training examples $(x_i, y_i) \in \mathbb{R}^d \times \{-1, 1\}$. To train a classifier on this input we start with a hard-margin variant of the ℓ_1-norm-regularized SVM Mangasarian (1999):

$$\min_{w \in \mathbb{R}^d, \gamma \in \mathbb{R}} \|w\|_1 \text{ s.t. } y_i(w^T x_i - \gamma) \geqslant 1\ \forall\ i.$$

Define matrix $A^+ \in \mathbb{R}^{d \times m}$ that has the positive examples in the columns, and $A^- \in \mathbb{R}^{d \times n}$ has the negative examples in the columns. Bennett and Bredensteiner (2000) provide a dual to the *soft-margin* ℓ_1-norm-regularized SVM with hinge loss, related to polytope distance formulations of the ℓ_∞-norm-regularized SVM Gärtner and Jaggi (2009):

$$\min_{u \in \Delta_m, v \in \Delta_n} \|A^+ u - A^- v\|_\infty \text{ s.t. } 0 \leqslant u, v \leqslant \frac{1}{R}. \tag{4.21}$$

The elementwise upper bounds on u and v and the parameter R come from the reduced convex hull construction in Bennett and Bredensteiner (2000). When $R = 1$, this yields a dual to the hard-margin problem. In the hard-margin case, the feasible set is $D := \Delta_m \times \Delta_n$, where Δ_m and Δ_n denote the unit simplices in $\mathbb{R}^m$ and $\mathbb{R}^n$ respectively, and the feasible set is a subset of D in the soft-margin case. Therefore, using Corollary 4.13, we get $C_f(\epsilon) \leqslant \frac{2}{\epsilon}\left(\operatorname{diam}_\infty(A^+\Delta_m) + \operatorname{diam}_\infty(A^-\Delta_n)\right)$.

Frank Wolfe Subproblems. For the dual problem in (4.21), express our objective as $f(x) := \|Ax\|_\infty$, for A and x such that $Ax = A^+u - A^-v$. We can then write f as

in (4.11), where the linear functions are given by the rows of A and the rows of $-A$. Define the approximate subdifferential as in (4.17), and in this case,

$$V(\hat{x}, \epsilon) = \{e_i \mid \hat{x}_i \geqslant \|\hat{x}\|_\infty - 2\epsilon\} \cup \{-e_i \mid \hat{x}_i \leqslant -\|\hat{x}\|_\infty + 2\epsilon\}.$$

We now turn to the subproblems in (4.3) for this $T(x, \epsilon)$ and feasible region D. Observe that the maximum over d will have a solution equal to a vertex of $T(x, \epsilon)$. An optimal z is found by:

$$\min_{z \in \mathbb{R}^{m+n},\ \mu \in \mathbb{R}} \mu \quad \text{s.t.} \quad \langle Az - Ax, d_v \rangle \leqslant \mu \quad \text{for } d_v \in V(Ax, \epsilon), \mathbf{1}^T z_+ = 1, \mathbf{1}^T z_- = 1, 0 \leqslant z \leqslant \frac{1}{R}, \tag{4.22}$$

where z_+ and z_- are the indices of the positive and negative examples respectively. There will be an optimal (z, μ) such that $n + m + 1$ linearly independent constraints are active, and $z_i \leqslant \frac{1}{R}$ will be active for no more than $2\lceil R \rceil$ indices i. This means that there is a solution z with no more than $2\lceil R \rceil + |V(Ax, \epsilon)| - 1$ nonzeros. Clearly, since the number of nonzero entries is strictly less than the total number of dimensions, the algorithm is sublinear.

Discussion. Here we note that the construction of coreset depends on the assumption that each iteration involves only $O(1)$ examples. This is true under the conditions which we describe now. For datasets that have well separated classes (small R), we have the following observation.

Observation 2. *Due to complementary slackness, for small ϵ we expect that $V(Ax, \epsilon)$ at the optimum will correspond to the nonzero features in the sparse separating hyperplane.*

Observation 2 provides a central intuition behind the use of coresets. When a problem is "easy" in a geometric or AI sense, having well-separated classes with few relevant features, the resulting coreset is small. Conversely, in problems that are "hard" the coreset size may approach the size of the original dataset. We also note that the steps, s_k may not be at a vertex of D, slightly complicating the coreset construction shown empirically in the later section. However, we *still* see useful results for nonsmooth f and polyhedral D if s is on a low-dimensional face of D, then s is a convex combination of a small number of vertices of D. Then the step will introduce more than one âŁœatom,âŁž indeed all of the vertices of the lowest-dimensional face on which s lies, but still a small number.

Tractability of subproblems. If we look at the subproblems from a Computational Geometry perspective, our algorithm will look roughly like Gilbert's algorithm Gilbert (1966). This is similar to what is described in Bennett and Bredensteiner (2000), though the ℓ_1-SVM case is not developed fully in that paper and the $T(x, e)$ construction introduces many additional issues. We see that ℓ_1-SVM subproblems i.e., (4.22) is solved in $O(R|V(Ax, e)|)$ time, given which example points determine the axis-aligned bounding box of the R-reduced convex hull of each class. The bounding box can be computed in time $O(Rnd)$ and $O(Rd)$ space: a cheap one-time cost performed before the optimization. One can also construct the bounding box lazily in $O(Rnd^*)$ time for d^* the cardinality of the union of all $V(Ax, e)$ across all iterations, and for problems that are easy in a geometric sense, we will have $d^* \ll d$ making the procedure sublinear. Hence, it is important to see that one would *not* necessarily directly perform the optimization in (4.3) using a generic solver.

Coreset Bound. The above results for ℓ_1-norm SVM provide the necessary lemmas to show a *coreset* bound for this problem. The construction of the coreset in this work follows the same intuition as when using the ordinary Frank Wolfe method Clarkson (2008). For each iterate $x_k \in \mathbb{R}^n$, and the corresponding subproblem solutions s_k, each index into the vector represents an input *example*. We may take those examples for which the index in s_k is nonzero, and call this set S_k. Using Algorithm 6 to build a coreset, we take the coreset at iteration K to be $\bar{S}_K = \bigcup_{k=1}^{K} S_k$.

Suppose we seek a coreset that will provide a $\epsilon-$ approximation solution. By Theorem 4.4, if we choose K to ensure that both sides of (4.5) are (multiplicatively) bounded above by $(1 + \epsilon)$, then the union $\bar{S}_K$ will provide this coreset. This approximation bound will be satisfied for

$$K \geqslant \frac{(2^{5/2}L + 2^{3/2}D_f)^2}{(1+\epsilon)^2} - 2.$$

The set of examples $\bar{S}_K$ will then be a $(1+\epsilon)$ coreset for any K satisfying this bound. We know from the above description that the ℓ_1-SVM dual objective will be $\|A\|-$Lipschitz, and that $C_f(\epsilon) \leqslant \frac{D_f}{\epsilon}$ for $D_f = 2\left(\text{diam}_\infty(A^+\Delta_m) + \text{diam}_\infty(A^-\Delta_n)\right)$.

For a coreset to be empirically useful, we also want it to be of small cardinality. The next step is therefore to bound $|\bar{S}_K|$. We earlier showed that $|S_k| \leqslant 2\lceil R \rceil + d^* - 1$

for all k. Using a union bound,

$$|\bar{S}_K| \leqslant K\left(2\lceil R\rceil + d^* - 1\right).$$

Combining these, we can use Algorithm 6 to construct a ϵ coreset of size $O\left(\frac{1}{\epsilon^2}\right)$.

4.4.1.2 Balanced development.

A direct application of the piecewise linear discussion in Section 4.1 is the problem of *balanced development*, see Nesterov (2009). Let $A = (a_1, ..., a_N) \in \mathbb{R}^{m\times n}$ where m is the number of attributes and n is the number of products. Hence, the entry A_{ij} is the *concentration* of attribute i in product j. Let b_i be the minimum concentration of attribute i that we require in our solution. Also, $p_j \geqslant 0$ is the unitary price of product j. The problem is then to make an investment decision between the products to maximize the minimum amount of each attribute. In Nesterov (2009), the author provides a dual problem:

$$\tau^* = \min_y\{\phi(y) : y \in \Delta_m\} \text{ where } \phi(y) = \max_{1\leqslant j\leqslant n} \frac{1}{p_j}\langle a_j, Ey\rangle,$$

$E \in \mathbb{R}^{m\times m}$ is diagonal with $E_{ii} = \frac{1}{b_i}$. Corollary 4.13 implies $C_f(\epsilon) \leqslant \frac{2}{\epsilon}(\text{diam}_\infty(A\Delta_m))^2$, and combined with the simplicial feasible region this provides a coreset over the attributes.

4.4.2 Multiway Graph Cuts

We consider the model for multi-label graph cuts in Călinescu et al. (1998); Niu et al. (2011). This problem seeks to assign one of d labels to each of n nodes. The graph structure is determined by similarity matrix W, with weight w_{uv} between vertices u and v. A convex relaxation of this problem is given by,

$$\min_x \quad \sum_{u\sim v} w_{uv}\|x_u - x_v\|_1 \quad \text{s.t.} \quad x_v \in \Delta_d \text{ for } v = 1, ..., n.$$

We further constrain a set of "seed" nodes to have $x_v = e_i$ if the label of seed v is $i \in \{1, ..., d\}$. Let $X \in \mathbb{R}^{d\times n}$ be the decision variable, a matrix such that each column is the soft labeling for a node. If we let B be the incidence matrix for an orientation of

the graph, and I the $d \times d$ identity matrix, then we can rewrite the objective as,

$$f(\mathrm{vec}(X)) = \|(B \otimes I)\mathrm{vec}(X)\|_1 = \|\mathrm{vec}(XB^T)\|_1,$$

where $\otimes$ is the matrix Kronecker product as in Section 1.4.

Now define $\hat{f}(\hat{x}) := \|\hat{x}\|_1$ so that $f(\mathrm{vec}(X)) = \hat{f}((B \otimes I)\mathrm{vec}(X))$. We start by deriving the subdifferentials for $\hat{f}$. For the approximate subdifferentials of $\hat{f}$ we choose the neighborhoods $\hat{N}(\hat{x}, \epsilon)$ to be the ℓ_1- ball of radius ϵ at x.

Lemma 4.14. *Similar to Lemma 4.10, the subdifferential of the neighborhood is a subset of the approximate differential, that is, if* $\hat{y} \in \hat{N}(\hat{x}, \epsilon)$, *then* $\partial \hat{f}(\hat{y}) \subseteq \hat{T}(\hat{x}, \epsilon)$.

Proof. If $\hat{f}$ is nondifferentiable on $\hat{N}(\hat{x}, \epsilon)$, the result follows from the definition.

Otherwise, if $\hat{f}$ is differentiable at all points on $\hat{N}(\hat{x}, \epsilon)$, then $|\hat{x}_i| > \epsilon$ for all i. Then, because $\|\hat{y} - \hat{x}\|_1 \leqslant \epsilon$, $\mathrm{sign}(\hat{y}_i) = \mathrm{sign}(\hat{x}_i)$ for all i, and $\nabla \hat{f}(\hat{y}) = \nabla \hat{f}(\hat{x})$ and we have the Lemma statement with equality. □

Theorem 4.15. *Consider any* $\hat{x}, \hat{s}$ *and* $\alpha \in [0, 1]$. *Let* $\hat{y} := \hat{x} + \alpha(\hat{s} - \hat{x})$. *Then,*

$$\min_{\hat{d} \in \hat{T}(\hat{x}, \epsilon)} \frac{1}{\alpha^2} \left(\hat{f}(\hat{y}) - \hat{f}(\hat{x}) - \langle \hat{y} - \hat{x}, \hat{d} \rangle \right) \leqslant \frac{2}{\epsilon} \|\hat{s} - \hat{x}\|_1. \tag{4.23}$$

Proof. We verify two cases,

Case 1. $\hat{y} \in \hat{N}(\hat{x}, \epsilon)$.

If we define the subgradient $\hat{d}$ elementwise by $\hat{d}_i = \mathrm{sign}(\hat{y}_i)$, then $\hat{d} \in \partial \hat{f}(\hat{y})$. Observe that $\langle \hat{y}, \hat{d} \rangle = \|y\|_1$. Then the left hand side of (4.23) can be bounded above as follows,

$$\hat{f}(\hat{y}) - \hat{f}(\hat{x}) - \langle \hat{y} - \hat{x}, \hat{d} \rangle = \|\hat{y}\|_1 - \|\hat{x}\|_1 - \|\hat{y}\|_1 + \langle \hat{x}, \hat{d} \rangle = \langle \hat{x}, \hat{d} \rangle - \|\hat{x}\|_1. \tag{4.24}$$

And because $\|\hat{d}\|_\infty \leqslant 1$, (4.24) is nonpositive. We therefore only need to bound the second case, next.

Case 2. $\hat{y} \notin \hat{N}(\hat{x}, \epsilon)$.

Here we have the lower bound on α that

$$\alpha \geqslant \frac{\epsilon}{\|\hat{s} - \hat{x}\|_1}, \tag{4.25}$$

and because $\|\hat{d}\|_\infty \leqslant 1$,

$$\begin{aligned}\hat{f}(\hat{y}) - \hat{f}(\hat{x}) - \langle \hat{y} - \hat{x}, \hat{d} \rangle &\leqslant \|\hat{y}\|_1 - \|\hat{x}\|_1 + \|\hat{y} - \hat{x}\|_1 \\ &= \|\hat{x} + \alpha(\hat{s} - \hat{x})\|_1 - \|\hat{x}\|_1 + \alpha\|\hat{s} - \hat{x}\|_1 \\ &\leqslant 2\alpha\|\hat{s} - \hat{x}\|_1,\end{aligned}$$

using the triangle inequality. The theorem statement then follows from this inequality and the bound on α. □

Remark 4.16. *Similar to Remark 4.12, the lower bound of α in equation (4.25) has no consequence for implementation.*

So, much like in the piecewise linear case, we can conclude that,

Corollary 4.17. *Given the problem of minimizing a piecewise linear function $f(x) = \|Ax + b\|_1$ over a bounded set D, the corresponding C_f is bounded above by $C_f(\epsilon) \leqslant \frac{2}{\epsilon}(\mathrm{diam}_1(AD))^2$.*

Note that in the case of multiway graph cuts, D consists of the Cartesian product of n simplices of dimension d. We therefore expect the bound from Corollary 4.17 will be $\Omega(n)$. The tractability, coreset (and hence the sublinearity) discussion in this case follows directly from the ℓ_1-regularized SVM case.

4.4.3 1-median

We can express the 1-median problem as an optimization over the simplex of convex combinations of the input points. Suppose we are given a (multi-)set of points $\mathcal{P} = \{p_1, ..., p_n\} \subset \mathbb{R}^d$. Let $A \in \mathbb{R}^{d \times n}$ be a matrix with p_i in column i. We can then write the 1-median problem as:

$$\min_{x \in \Delta_n} f(x) := \hat{f}(Ax) := \frac{1}{n}\sum_{i=1}^{n} \hat{f}_i(Ax) := \frac{1}{n}\sum_{i=1}^{n} \|Ax - p_i\|_2. \tag{4.26}$$

The notation $\|\cdot\|$ denotes the ℓ_2 norm. Here, f is nondifferentiable for any x such that $Ax \in \mathcal{P}$, and differentiable elsewhere. We show that $C_f(\epsilon) \leqslant \frac{2}{\epsilon}\mathrm{diam}_2(A\Delta_n)^2$, and extend this to a coreset result. We note that recently Cohen et al. (2016) developed a Stochastic Gradient Descent algorithm that has a running time of $O(1/\epsilon^2)$

matching our results. The key difference is that the algorithm in Cohen et al. (2016) is randomized, so their result holds only in probability < 1.

The subdifferentials are given by,

$$\partial f(x) = A^T \partial \hat{f}(Ax) = \frac{1}{n} A^T \sum_{i=1}^{n} \partial \hat{f}_i(Ax), \quad \partial \hat{f}_i(Ax) = \begin{cases} \{\frac{Ax - p_i}{\|Ax - p_i\|}\} & \text{if } Ax \neq p_i, \\ \bar{\mathcal{B}}(0,1) & \text{if } Ax = p_i. \end{cases}$$

Note that the $\frac{1}{n}$ factor in (4.26) is necessary to produce sensible approximation bounds. If we duplicate each point in our input set, this will double the value of the sum in the objective for any choice of median. Without the $\frac{1}{n}$ factor, any approximation bound expressed in terms of the objective would necessarily be $\Omega(n)$.

The k-median problem has been considered previously in the coreset setting, Har-Peled and Kushal (2005) give a coreset construction by binning the points to derive a coreset of size $O(k\epsilon^{-d} \log n)$. For $k > 1$, the k-median problem is known to be nonconvex, so we here consider the problem of calculating a single median.

For this problem, we choose neighborhoods as,

$$N(x, \epsilon) = \{u \mid \|Au - Ax\| < \epsilon \text{ and } u - x \in \operatorname{im}(A^T)\}. \tag{4.27}$$

Note that the choice of $N(x, \epsilon)$ in (4.27) meets the assumptions in the convergence bounds above, as $x \in N(x, \epsilon)$, and $N(x, \epsilon) \subseteq B(x, \|A^\dagger\|\epsilon)$. $\|A^\dagger\|$ is the operator norm of the pseudoinverse. The subdifferentials are described by,

$$\partial f(x) = \frac{1}{n} A^T \sum_{i=1}^{n} \partial \hat{f}_i(Ax), \quad \partial \hat{f}_i(Ax) = \begin{cases} \{\frac{Ax - p_i}{\|Ax - p_i\|}\} & \text{if } Ax \neq p_i, \\ \bar{\mathcal{B}}(0,1) & \text{if } Ax = p_i. \end{cases}$$

4.4.4 Iteration bounds

In order to bound $C_f(\epsilon)$, we bound the quantity:

$$\frac{1}{n\alpha^2} \min_{u \in N(x,\epsilon)} \left(\sum_{Au \neq p_i} \left(\|Ay - p_i\| - \|Ax - p_i\| - \left\langle Ay - Ax, \frac{Au - p_i}{\|Au - p_i\|} \right\rangle \right) \right.$$
$$\left. + E(u)(\|Ay - Au\| - \|Ax - Au\| - \|Ay - Ax\|) \right),$$

for any x, s in the simplex and $y = x + \alpha(s - x)$ for $\alpha \in [0, 1]$. $E(u)$ is the number of points such that $Au = p_i$. This is equivalent to the terms in the supremum in $C_f(\epsilon)$, where we have substituted the analytic solution for the minimum element of $\partial f(u)$. By the triangle inequality, for any u,

$$\|Ax - Au\| + \|Ay - Ax\| \geqslant \|(Ax - Au) + (Ay - Ax)\| = \|Ay - Au\|.$$

Since the nondifferentiable terms are nonpositive, we only need to bound the differentiable terms. First, assume that $y - x \in \mathrm{im}(A^T)$. This is without loss of generality, as we can replace s with its projection $s' \in x + \mathrm{im}(A^T)$ and y with $y' = x + \alpha(s' - x)$. Because $s' - s \in \ker(A)$, this yields $Ay' = Ay$, and the quantity we seek to bound will be identical.

Case 1. $y \in N(x, \epsilon)$.

If $Au = Ay$, the i^{th} differentiable term is given by,

$$\begin{aligned}
&\|Ay - p_i\| - \|Ax - p_i\| - \left\langle Ay - Ax, \frac{Ay - p_i}{\|Ay - p_i\|} \right\rangle \\
&= \frac{1}{\|Ay - p_i\|} \left(\langle Ay - p_i, Ay - p_i \rangle - \|Ax - p_i\| \|Ay - p_i\| - \langle Ay - Ax, Ay - p_i \rangle \right) \\
&= \frac{1}{\|Ay - p_i\|} \left(\langle Ax - p_i, Ay - p_i \rangle - \|Ax - p_i\| \|Ay - p_i\| \right),
\end{aligned}$$

which by the Cauchy-Schwarz inequality is nonpositive. Therefore, in order to provide an upper bound on $C_f(\epsilon)$ we need to only consider the second case below.

Case 2. $y \notin N(x, \epsilon)$.

First, observe that in this case we have a lower bound on α. Specifically, because y lies outside the neighborhood around x defined by (4.27) (and in the set $x + \mathrm{im}(A^T)$ by assumption), $\|Ay - Ax\| = \alpha \|As - Ax\| \geqslant \epsilon$,

$$\alpha \geqslant \frac{\epsilon}{\|As - Ax\|} \geqslant \frac{\epsilon}{\mathrm{diam}(A\Delta_n)}. \tag{4.28}$$

Consider in the following, any choice of u from the neighborhood around x.

Plugging in $y = x + \alpha(s - x)$ to the differentiable terms,

$$\begin{aligned}
&\frac{\sum_{Au \neq p_i} \|Ay - p_i\| - \|Ax - p_i\| - \left\langle Ay - Ax, \frac{Ax - p_i}{\|Ax - p_i\|} \right\rangle}{n\alpha^2} \\
&= \frac{\begin{array}{l}\sum_{Au \neq p_i} \|Ax - p_i + \alpha(As - Ax)\| \\ \quad - \|Ax - p_i\| - \alpha \left\langle As - Ax, \frac{Au - p_i}{\|Au - p_i\|} \right\rangle\end{array}}{n\alpha^2} \\
&\leqslant \frac{1}{n\alpha^2} \sum_{Au \neq p_i} 2\alpha \|As - Ax\| \\
&\leqslant \frac{2\|As - Ax\|}{\alpha} \\
&\leqslant \frac{2\mathrm{diam}(A\Delta_n)}{\alpha},
\end{aligned} \tag{4.29}$$

where we used the triangle and Cauchy-Schwarz inequalities for the inequalities. The combination of (4.28) and (4.29), taken over all x, s and all α that fall in this case, gives:

$$C_f(\epsilon) \leqslant \frac{2}{\epsilon}\mathrm{diam}(A\Delta_n)^2.$$

Next, observe that if ϵ is small, as will be the case throughout the optimization if we choose $\epsilon_k \in \Theta(\sqrt{\alpha_k})$ with a small constant term, then with high probability the neighborhood around the iterates for any k that are not near the solution will be differentiable. This will yield $T(x, \epsilon) = \{\nabla f(x)\}$ and the subproblems will be a linear program over the simplex, which will only introduce no more than one nonzero as in Clarkson (2008) which indeed means that the procedure is sublinear.

4.4.5 Sparse PCA with constraints

Sparse PCA is an important problem that is widely used for feature extraction and learning. In this problem we consider the formulation given in Grbovic et al. (2012). The objective of sparse PCA is to decompose a covariance matrix S into near orthogonal principal components $[u_1, ..., u_k]$, while constraining the number of nonzero components of each u_k to a required constant $r \in \mathbb{N}$. Hence the problem of maximizing the variance of component u_k with the cardinality constraint can be written

as,

$$\max_{u} \quad u^T S u \quad \text{s.t.} \quad \|u\|_2 = 1, \ \ \text{card}(u) \leqslant r.$$

Writing down the convex relaxation of the problem after setting $U = uu^T$, we get the following optimization problem,

$$\max_{U} \quad \text{Tr}(SU) \quad \text{s.t.} \quad \text{Tr}(U) = 1, \ \ 1^T|U|1 \leqslant r, \ \ U = U^T, \ \ U \succeq 0, \ \ U \in \mathbb{R}^{n \times n} \tag{4.30}$$

where Tr denotes the trace operator as defined in Section 1.4. The last inequality constraint in (4.30) is the standard Linear Matrix Inequality that requires U to be positive semidefinite. Note that we have exponential number of linear constraints in the form of $1^T|U|1 \leqslant r$ constraint. Hence, in general, this problem might practically take very long to solve unless one uses a specialized interior point method that exploits the structure of the constraints. Hence, these constraints are taken to the objective with a penalty ρ associated with it as,

$$\max_{U} \quad \text{Tr}(SU) - \|\rho \text{vec}(U)\|_1 \ \ \text{s.t.} \quad \text{Tr}(U) = 1, \ \ U = U^T, \ \ U \succeq 0, \ \ U \in \mathbb{R}^{n \times n},$$

where $\text{vec}(U)$ denotes vectorized form of U. To this formulation, Grbovic et al. (2012) suggests adding a class of feature grouping constraints that is motivated by the maintenance scheduling problem Cui (2008). Let us denote by $\mathbf{1}$, a reliability vector such that $\mathbf{1}_i \in [0, 1)$ is a probability that sensor i will need maintenance during a certain time period. Then, the *reliability* matrix L is constructed by setting each column to $\mathbf{1}$ and then setting $L_{ii} = 0 \ \forall \ 1 \leqslant i \leqslant n$. Using similar techniques we end up with the following formulation,

$$\begin{aligned} &\max_{U} \ \ \text{Tr}(SU) - \|\rho \text{vec}(U)\|_1 - \|\text{vec}(\eta U \circ \log(1 - L))\|_1 \\ &\text{s.t.} \ \ \text{Tr}(U) = 1, U = U^T, U \succeq 0, U \in \mathbb{R}^{n \times n}, \end{aligned}$$

where η is the penalty that controls the reliability of the component and $\circ$ is the standard Hadamard product or elementwise product. Let $\vec{U}$ denote the vectorized form of U for simplicity in notation and similarly $\text{vec}(\log(1 - L)) = \vec{l}$. Note that

$\vec{l}, \vec{U} \in \mathbb{R}^{n^2 \times 1}$. Then the problem can be written as,

$$\max_{U} \quad \text{Tr}(SU) - \sum_{i=1}^{n^2} \left(\rho + \eta \vec{l}_i\right) |\vec{U}_i| \quad \text{s.t.} \quad \text{Tr}(U) = 1,\ U = U^T,\ U \succeq 0,\ U \in \mathbb{R}^{n \times n}.$$

Noting that $\text{Tr}(SU) = \sum_{i=1}^{n} \sum_{j=1}^{n} S_{ij} U_{ij} = \vec{S}^T \vec{U} = \sum_{i=1}^{n^2} \vec{S}_i \vec{U}_i$, we can write the objective function as a supremum of 2^{n^2} affine functions. Also using the Sion's min-max theorem (Komiya, 1988), we can convert this to a minimization problem. Hence the problem becomes,

$$\min_{U} \quad \max_{i} \left(a_i^T \vec{U}\right) \quad \text{s.t.} \quad \text{Tr}(U) = 1,\ U = U^T,\ U \succeq 0,\ U \in \mathbb{R}^{n \times n}.$$

Let $S := \{U : \text{Tr}(U) = 1,\ U = U^T,\ U \succeq 0,\ U \in \mathbb{R}^{n \times n}\}$.

Lemma 4.18. *The feasible set of sparse PCA with constraints problem, S is bounded.*

Proof. We know that $U \succeq 0,\ U = U^T \iff \lambda_i \geqslant 0$ where λ_is are the eigenvalues of U and $\text{Tr}(U) = 1 \iff \sum_i \lambda_i = 1$. Let $S_\lambda := \{\lambda : \sum_i \lambda_i = 1,\ \lambda \geqslant 0,\ \lambda \in \mathbb{R}^n\}$. Hence S is bounded if and only if S_λ is bounded. But S_λ is trivially bounded since it is the simplex. □

Let $A \in \mathbb{R}^{2^{n^2} \times n^2}$ be the matrix containing all the a_is stacked on top of each other. Hence our optimization problem can be written as,

$$\min_{U} \quad \max_{i} \left(A\vec{U}\right)_i \quad \text{s.t.} \quad U \in S.$$

We can now define the neighborhoods in a similar way as Section 4.4.1,

$$\hat{N}\left(\hat{\vec{U}}, \epsilon\right) = \{\hat{\vec{X}} \mid \|\hat{\vec{U}} - \hat{\vec{X}}\|_\infty \leqslant \epsilon\};$$
$$N\left(\vec{U}, \epsilon\right) = \{\vec{X} \mid \|A\vec{X} - A\vec{U}\|_\infty \leqslant \epsilon \text{ and } \vec{X} - \vec{U} \in \text{im}\left(A^T\right)\},$$

and define ,

$$V\left(\hat{\vec{U}}, \epsilon\right) = \left\{e_i \mid \hat{\vec{U}}_i \geqslant \left(\max_j \hat{\vec{U}}_j\right) - 2\epsilon\right\}.$$

With these constructions, similar results as in Section 4.4.1 are obtained. Note that unlike the other cases, the subproblems here are not as easy to solve and we might

require advanced numerical optimization solvers to solve the subproblems efficiently in practice.

4.4.6 Generic Subproblems

As we saw in the above sections, specialized algorithms can be used to solve the subproblems that are induced by (4.3) for many important applications. But occasionally, we might encounter problems that do not possess an inherent structure that can be easily exploited; for these cases, we now provide a generic method (or algorithm) for solving the subproblems. Specifically, we will consider the case when the optimization in (4.3) is over a union of convex sets $C = \cup_{i=1}^{n} C_i$ where C_i is a convex set $\forall\ i = 1, ..., n$. Recall that the subdifferential of a proper convex function is a compact set for all the points in the interior of the domain Robinson (2012), hence, many more problems (that were not considered in Section 4.4) fall under this case, for example, piecewise quadratic optimization that is commonly used in economic load dispatching problems, see Lee and Kim (2002). Assuming that we have access to the zeroth and first order oracles, C can be explicitly described by the inequalities $g_{ij}(x) \leqslant 0$ where g_{ij} are convex functions $\forall\ i, j$ and the total set of constraints has finite cardinality. It is easy to show that the problem of optimizing a convex function g over C can be formulated as the following convex optimization problem,

$$\min_{x} \quad g(x) \quad \text{s.t.} \quad s_i g_{ij}\left(z_i/s_i\right) \leqslant 0, \forall i, j,\ 1^T s = 1, s \geqslant 0, x = \sum_{i=1}^{n} z_i. \tag{4.31}$$

There are two advantages of using the reformulation in (4.31) instead of solving n different convex problems: (1) in certain cases, it might be easy to design algorithms to get approximate solutions much more efficiently Garber and Hazan (2011) and (2) standard solvers are actively being developed to handle perspective reformulations as described in Günlük and Linderoth (2012). From this discussion, we can conclude that the subproblems can be solved in polynomial time.

4.5 Experiments

In this section, we present experiments to assess our convergence rate numerically and provide some practical intuition for the bounds obtained in the sections above.

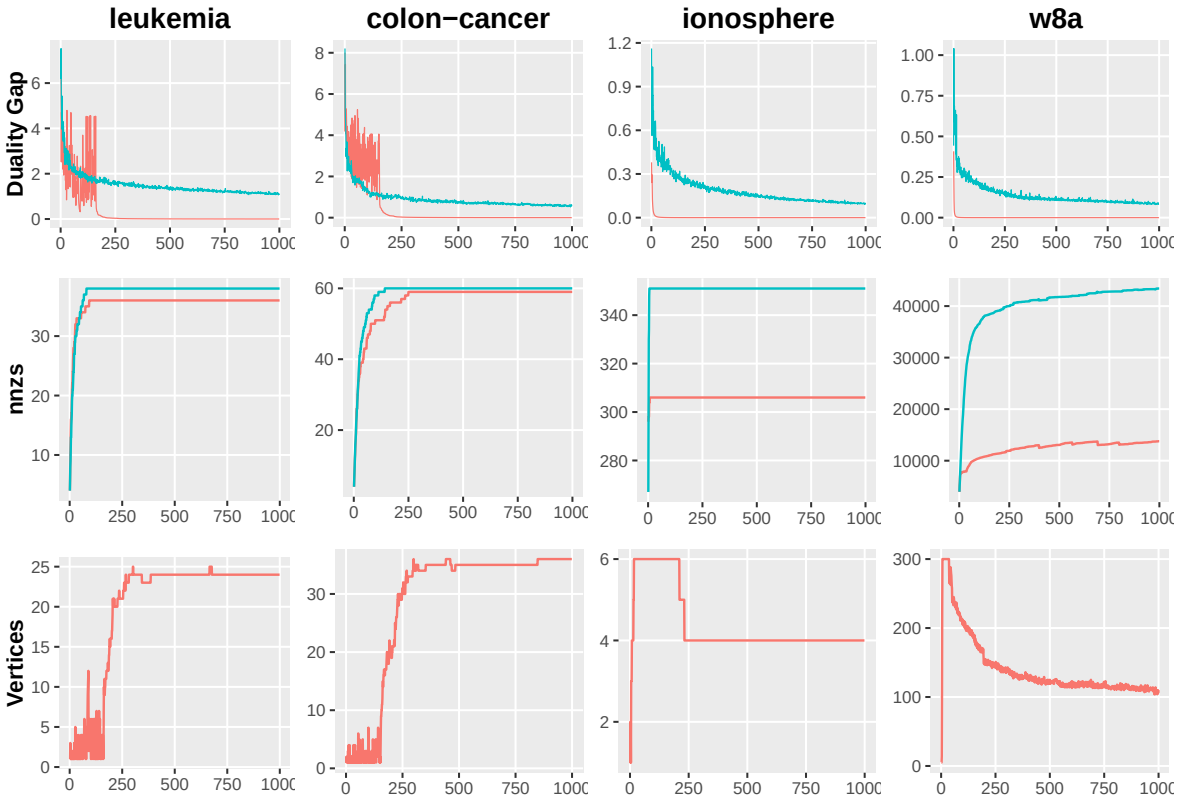


Figure 4.1: Convergence results for Alg. 6 (red) and randomized smoothing of Lan (2013) (blue) on the SVM datasets. The horizontal axis denotes the iterations.

We provide the running time whenever it is significant (more than a few seconds). To keep the extraneous effects of specialized libraries and the number of processors negligible, all our subproblems were solved using a generic Linear Programming solver on a single core. The results show that the algorithm presented here is not only practical but can also be made competitive using simple alternative standard techniques to solve the subproblems.

4.5.1 Support Vector Machines with ℓ_1 regularization

We first demonstrate results of our algorithm for ℓ_1-regularized SVM on a collection of four test datasets provided by the authors of Yuan et al. (2010) (see Figure 4.1).

For this evaluation, we also compared our method with the randomized smoothing method of Lan (2013) shown in blue in Fig. 4.1. The top row of plots in Figure 4.1 (y−axis showing the duality gap) suggest that on all four datasets (leukemia, colon-cancer, ionosphere, w8a), the convergence of the baseline is slower as a function of

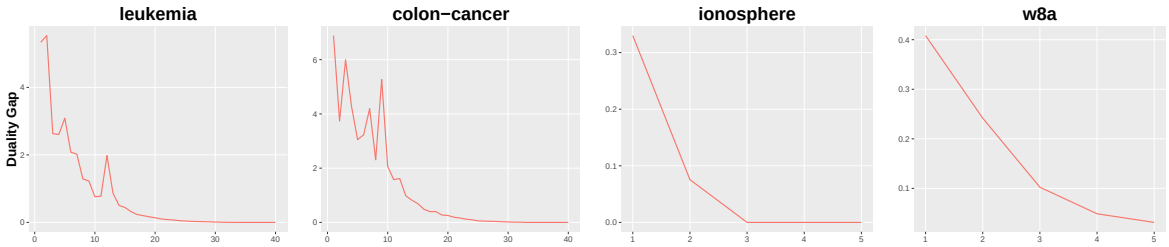


Figure 4.2: Convergence results for Alg. 6 using bisection line search.

the number of iterations. The second and the third row show that the coreset size and the vertices in $|V(Ax, \epsilon)|$ increase *at most linearly* with the number of iterations as predicted in Section (4.4.1.1). We see that the number of vertices in $|V(Ax, \epsilon)|$ remains *very small* throughout our experiments. Since $\alpha_k = 2/(k+2)$ in Fig. 4.1 (which is agnostic of f, D), we observe a little erratic behavior of our algorithm with respect to the duality gap and number of vertices in the earlier iterations. While the convergence results shown for Lan (2013) are, in expectation, of the same order as the ones we show here, the bound is quite loose for these problems and we find that the practical convergence rate can be slow. This is especially important as the computation time of each iteration of that method in Lan (2013) increases with the iteration count, as more samples of the gradient are drawn in later iterations (unlike our algorithm). Moreover, Figure 4.2 shows that if we use a bisection line search instead of the step size policy suggested in the proof of Theorem 4.4, our algorithm converges (measured using the duality gap) in fewer than 40 iterations making it practically fast while also keeping our theoretical results intact.

4.5.2 The 1-median problem

We tested a simple Frank Wolfe based solver for the 1-median problem on synthetic data (similar to existing works for this problem). A set of points of size n is sampled from a multivariate random Normal distribution. To keep the presentation succinct, we discuss those parts of this experiment that demonstrate additional useful properties of the algorithm beyond those described for ℓ_1-SVM. The left plot in the top row of Figure 4.3 shows how the size of the coreset varies as we increase n whereas the right plot shows the number of iterations required for convergence. Importantly, the size of the coreset is *independent* of n, as the problem we are solving is fundamentally no harder. The second row shows the computational time taken: left plot shows the

total time taken whereas the right plot shows the time taken per iteration. We see that both of these scale sublinearly with the number of points. The plots agree with the theoretical results from earlier sections concerning the size of coreset and number of iterations required to obtain a solution.

4.5.3 Multiway Graph Cuts

In order to demonstrate that our algorithm is scalable to large datasets and hence it is practically applicable, we tested our algorithm on the *dblife* dataset Niu et al. (2011) which consists of 9168 labels. Therefore, D is the product of n 9168-dimensional simplices. Note that this problem consist of *approximately 20 million decision variables* and has been tackled via distributed schemes implemented on a cluster Niu et al. (2011). Due to the large of number of labels, this dataset is ideal for evaluating our algorithm as the analysis implies that the number of labels chosen must *increase linearly with the number of iterations*. This means that we get a sparse labeling scheme, i.e., the *coreset here is the subset of the total number of labels*. We evaluated this behavior by using increasing subsets of the total number of labels. We find that even when the total number of labels is small; this trend becomes more prominent as progressively larger sets of labels are used, shown in Figure 4.3 (row 3).

Problem	C_f	Coreset
ℓ_1-SVM	$\frac{2}{\epsilon}(\text{diam}_\infty(A^+\Delta_m) + \text{diam}_\infty(A^-\Delta_m))$	Support vectors
1-median	$\frac{2}{\epsilon}\text{diam}_2(A\Delta_n)^2$	Points
Graph cuts	$\frac{2}{\epsilon}(\text{diam}_1(AD))^2$	Labels
Balanced Development	$\frac{2}{\epsilon}(\text{diam}_1(A\Delta_m))^2$	Attributes
Sparse PCA with constraints	$\frac{2}{\epsilon}(\text{diam}_\infty(AS))^2$	Features

Table 4.1: A summary of the relevant quantities derived for each problem described in the previous sections.

4.5.4 Interesting empirical behavior and its potential relation to existing work

The experiments highlight some important practical aspects of the proposed algorithm.

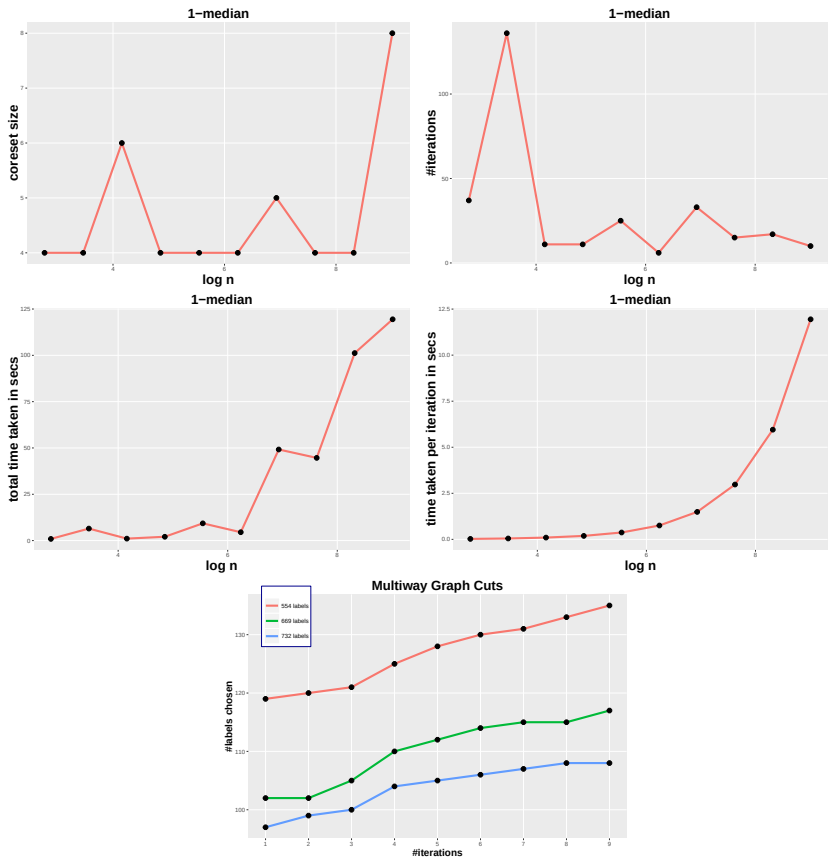


Figure 4.3: (Top row) Plotting the size of the coreset (left) and number of iterations (right) for 1-median as we vary the size n of a randomly sampled set of points. We iterate until achieving a primal-dual gap of 10^{-6}; 2^{nd} row shows the total time taken (left) and time taken per iteration (right); 3^{rd} row shows results for Graph cuts problem on dblife. Each LP subproblem took approximately 60 seconds using CPLEX solver.

1. **How often do we encounter poor subproblems?** The number of vertices in $|V(Ax, \epsilon)|$ given by our analysis seems to be an overestimate than what is practically observed. Assuming that we initialize at a point whose neighborhood does not contain any nonsmooth points, using the fact that the set of nondifferentiable points of a convex function on a compact set is of measure zero, we can show that the next iterate of the algorithm also does not contain any nonsmooth points in its neighborhood with high probability. But extending this argument to the next iterate becomes problematic. It seems like a Lovasz Local Lemma type result Kolmogorov (2015) will be able to shed some light on this behavior but the independence assumption is inapplicable, so one has to introduce randomness in the algorithm. But a separate sparsity analysis might

have to be done to obtain coreset results.

2. **Why does our algorithm converge so fast?** The empirical behavior of certain algorithms have been strongly associated with the *manifold identification property* in recent works, see Lee and Wright (2012). Our algorithm when using a bisection line search strategy as shown in Fig. 4.3, converges within a few iterations and is faster than the state of the art solvers for that problem. It will be interesting to see if our algorithm satisfies some variation or extension of the properties that are described in Lee and Wright (2012), but one issue is that the algorithm presented in Lee and Wright (2012) has randomization incorporated at each iteration making it hard to adapt that analysis for our algorithm.

3. **Tightness of convergence.** Let us consider the case of ℓ_1-SVM. Assume that the positive and negative labeled instances be generated by a Gaussian distribution with mean 0 and μ respectively and variance σI. Using the properties of maximum of sub-gaussians, we have that,

$$\mathbb{E}\left(\operatorname{diam}_\infty(A^+\Delta_m) + \operatorname{diam}_\infty(A^-\Delta_m)\right) \leqslant \sigma \log m + \mu + \sigma \log m = 2\sigma \log m + \mu,$$

where the expectation is taken over the randomness in the data: A^+, A^-. This shows that the dependence of C_f is logarithmic in the dimensions. Similarly we will get $\sqrt{n}, n \log n$ for ℓ_2, ℓ_1 norm diameters respectively showing that the $\mathbb{E}C_f$ is near linear in the dimensions. Now we plug this estimate in the convergence result in Theorem (4.4) to get,

$$\mathbb{E}\left(f(x_k) - f(x_*)\right) \leqslant \mathbb{E}\left(\frac{2^{5/2}L + 2^{3/2}D_f}{\sqrt{k+2}}\right) = 2^{5/2}\left(\frac{L + \mathbb{E}D_f}{\sqrt{k+2}}\right) = O\left(\frac{L + \log m}{\sqrt{k}}\right).$$

Hence we can see that by choosing an appropriate neighborhood depending on the application, we can get faster convergence. This aspect is either not present in other Frank Wolfe algorithms that solve nonsmooth problems or involve a looser dependence on the constants involved for example, Hazan and Kale (2012) achieve a $O\left(\frac{Lm}{\sqrt{k}}\right)$ convergence compared to our $O\left(\frac{L+\log^2 m}{\sqrt{k}}\right)$ here.

4.6 Discussion

We have presented novel coreset bounds and optimization schemes for nondifferentiable problems encountered in statistical machine learning and computer vision. The algorithm calculates sparse approximate solutions to the corresponding optimization problems deterministically in a number of iterations independent of the size of the input problem, depending only on the approximation factor and the sum of the Lipschitz constant and a nonlinearity term. The central result in Theorem 4.4 applies to any problem of the very general form in (4.1) with mild conditions, potentially suggesting a number of other applications beyond those considered here. Though a general condition to characterize *all* cases where the internal subproblem is efficiently solvable may not be available, we show a broad/useful class that *is* efficiently solvable.

Finally, we point out a few technical properties that differentiate our method from existing FW type methods for nonsmooth problems Argyriou et al. (2014); Pierucci et al. (2014).

1. Many methods rely on smoothing the objective function by a *proximal* function that needs to be designed piecemeal for specific problems. While examples exist where these functions can be readily derived, to our knowledge there are no standard recipes for deriving proximal functions in general.

2. Several existing methods assume that the proximal iteration can be solved efficiently. However, it is known that in general, the worst case complexity of a single proximal iteration is the same as solving the original optimization problem Parikh and Boyd (2014).

3. It is not yet clear how coreset results can be derived for many existing methods, or if it is even possible. For example, (especially) in the *very large scale* setting, coreset results enable practical applications where one can store a subset of the (training) dataset and still be able to perform nearly as well (on the test data). The basic expectation is that more data should *not* always make a problem computationally harder. An exciting implication behind coreset results in (Clarkson, 2008) is that this can be avoided in certain cases. But (Clarkson, 2008) assumes that f is smooth: an artifact of the optimization rather than a requirement of coresets per se — we showed that this assumption is not necessary. In (the next)

Chapter 5, we will see how CG algorithms perform while training nonconvex models.

In closing, while the algorithm may not be the defacto off-the-shelf option for *all* nonsmooth problems, for many problems it offers a very competitive (generally applicable) alternative, and in certain cases, the theory nicely translates into significant practical benefits as well.

Chapter 5

Explicitly Imposing Constraints in Deep Networks

In the previous chapters, we saw how CG algorithm in Algorithm 4 can be used to estimate motion between images in Chapter 3 and learn ML models with nonsmooth objective function in Chapter 4. However, these vision and ML tasks were formulated as a convex optimization problem. In this chapter, we analyze the stochastic form of CG algorithm with applications to training Deep Neural Networks under constraints that are explicitly imposed on the parameters of the model. We show extensive empirical evidence to support our claim that it is possible to efficiently compute *low complexity* optimal solutions using CG type methods in (Ravi et al., 2019b).

5.1 Introduction

Recall from Chapter 1 that Deep Neural Networks (DNNs) are over-parameterized models that are trained using first order methods. Here, the input data is often transformed/composed by a sequence of nonlinear functions in order to learn complex representations of the input data. The learning or fitting problem in DNNs in the supervised setting is often expressed as the following stochastic optimization problem,

$$\min_{W} \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}} \mathcal{L}(W;(x,y))$$

where $W = W_1 \times \cdots \times W_l$ denotes the Cartesian product of the weight matrices of the network with l layers that we seek to learn from the data (x, y) sampled

from the underlying distribution $\mathcal{D}$. Here, x can be thought of the "features" (or predictor variables) of the data and y denotes the "labels" (or the response variable). The variable W parameterizes the function that predicts the labels given the features whose accuracy is measured using the loss function $\mathcal{L}$. For simplicity, the specification above is intentionally agnostic of the activation function we use between the layers and the specific network architecture. Most common instantiations of the above task are non-convex but results in the last 5 years show that good minimizers can be found via first order methods such as SGD and its variants, see Section 2.1.2. Recent results have also explored the interplay between the overparameterization of the network, its degrees of freedom and issues related to global optimality Soudry and Carmon (2016).

5.1.1 Related Work

Independent of the architecture we choose to deploy for a given task, one may often want to impose additional constraints or regularizers, pertinent to the application domain of interest. In fact, the use of task specific constraints to improve the behavioral performance of *neural networks*, both from a computational and statistical perspective, has a long history dating back at least to the 1980s Platt and Barr (1988); Zhang and Constantinides (1992); Wahba et al. (1995). These ideas are being revisited Rudd et al. (2014) motivated by generalization, convergence or simply as a strategy for compression Cheng et al. (2017). However, using constraints on the types of architectures that are common in modern AI problems is still being actively researched by various groups. For example, Mikolov et al. (2014) demonstrated that training Recurrent Networks can be accelerated by constraining a part of the recurrent matrix to be close to identity. Sparsity and low-rank encouraging constraints have shown promise in a number of settings Tai et al. (2015). In an interesting paper, Pathak et al. (2015) showed that linear constraints on the output layer improves the accuracy on a semantic image segmentation task. Márquez-Neila et al. (2017) showed that hard constraints on the output layer yield competitive results on the pose estimation task and Oktay et al. (2017) used anatomical constraints for cardiac image analysis. In generative modeling, (Frerix et al., 2019) showed that linear inequality constraints may be imposed to restrict the output a variational autoencoder. This suggests that while there are some results demonstrating the value of *specific* constraints for *specific* problems, the development is still in a nascent stage. It is, therefore, not surprising

that the existing software libraries for deep learning (DL) offer little to no support for hard constraints. For example, Keras only offers support for simple bound constraints.

5.1.2 Optimization Schemes

Let us momentarily set aside the issue of constraints and discuss the choice of the optimization schemes that are in use today. There is little doubt that SGD algorithms dominate the landscape of DL problems in AI. Instead of evaluating the loss and the gradient over the full training set, SGD computes the gradient of the parameters using a few training examples. It mitigates the cost of running back propagation over the full dataset and comes with various guarantees as well Hardt et al. (2016). The reader will notice that part of the reason that constraints have not been intensively explored may have to do with the interplay between constraints and the SGD algorithm Márquez-Neila et al. (2017). While some regularizers and "local" constraints are easily handled within SGD, some others require a great deal of care and can adversely affect convergence and practical runtime Bengio (2012). There are also a broad range of constraints where SGD is *unlikely to work well based on theoretical results known today* — and it remains an open question in optimization Johnson and Zhang (2013). We note that algorithms other than the standard SGD have remained a constant focus of research in the community since they offer many theoretical advantages that can also be easily translated to practice Dauphin et al. (2015). These include adaptive subgradient methods such as Adagrad, the RMSprop algorithm, and various adaptive schemes for choosing learning rate including momentum based methods Goodfellow et al. (2016). However, notice that these methods only impose constraints in a "local" fashion since the computational cost of imposing global constraints using SGD-based methods becomes extremely high Pathak et al. (2015).

5.2 Do we need to impose constraints?

The question of why constraints are needed for statistical learning models in vision and machine learning can be restated in terms of the need for regularization while learning models. Recall that regularization schemes in one form or another go nearly as far back as the study of fitting models to observations of data Wahba (1990). Broadly speaking, such schemes can be divided into two related categories: algebraic and statistical. The **first** category may refer to problems that are otherwise not possible

or difficult to solve, also known as ill-posed problems Tikhonov et al. (1987); Wahba (1987). For example, without introducing some additional piece of information, it is not possible to solve a linear system of equations $Ax = b$ in which the number of observations (rows of A) is less than the number of degrees of freedom (columns of A). In the **second** category, one may use regularization as a way of "explaining" data using simple hypotheses rather than complex ones, for example, the minimum description length principle Rissanen (1985). The rationale is that, complex hypotheses are less likely to be accurate on the unobserved samples since we need more data to train complex models. Recent developments on the theoretical side of DL showed that imposing simple but *global* constraints on the parameter space is an effective way of analyzing the sample complexity and generalization error Neyshabur et al. (2015b). Hence we seek to solve,

$$\min_{W} \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}} \mathcal{L}(W;(x,y)) + \mu R(W) \tag{5.1}$$

where $R(\cdot)$ is a suitable regularization function for a fixed $\mu > 0$. We usually assume that $R(\cdot)$ is simple, in the sense that the gradient can be computed efficiently. Using the Lagrangian interpretation, Problem (5.1) is the same as the following constrained formulation,

$$\min_{W} \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}} \mathcal{L}(W;(x,y)) \text{ s.t. } R(W) \leqslant \lambda \tag{5.2}$$

where $\lambda > 0$. Note that when the loss function $\mathcal{L}$ is convex, both the above problems are equivalent in the sense that given $\mu > 0$ in (5.1), there exists a $\lambda > 0$ in (5.2) such that the *optimal solutions to both the problems coincide* (see Sec 1.2 in Bach et al. (2012a)). In practice, both λ and μ are chosen by standard procedures such as cross validation.

5.2.1 Finding Pareto Optimal Solutions

On the other hand, when the loss function is nonconvex as is typically the case in DL, formulation (5.2) is more powerful than (5.1). We state it as the following observation.

Observation 3. *For a fixed $\lambda > 0$, there might be solutions W^*_λ of (5.2) for which there exists no $\mu > 0$ such that $W^*_\lambda = W^*_\mu$ whereas any solution of problem (5.1) can be obtained for some λ in (5.2) (Section 4.7 in Boyd and Vandenberghe (2004a)).*

It turns out that it is easier to understand this phenomenon through Multiobjective Optimization (MO). In MO, care has to be taken to even define the notion of optimality of feasible points (let alone computing them efficiently). Among various notions of optimality, we will now argue that Pareto optimality is the most suited for our goal.

Recall that our goal is to find W's that achieve low training error and are at the same time "simple" (as measured by R). In this context, a Pareto optimal solution is a point W such that none of $\mathcal{L}(W)$ or $\mathsf{R}(W)$ in (5.2) or (5.1) can be made better without making the other worse, thus capturing the essence of overfitting effectively. In practice, there are many algorithms to find Pareto optimal solutions and this is where problem (5.2) dominates (5.1). Specifically, formulation (5.1) falls under the category of "scalarization" technique whereas (5.2) is ϵ-constrained technique. It is well known that when the problem is nonconvex, ϵ-constrained technique yields pareto optimal solutions whereas scalarization technique does not Boyd and Vandenberghe (2004a)!

Finally, we should note that even when the problems (5.1) and (5.2) are equivalent, in practice, algorithms that are used to solve them can be very different.

Our Results. We show that many interesting global constraints of interest in AI can be explicitly imposed using a classical technique, Conditional Gradient (CG) that has not been deployed much at all in deep learning. We analyze the theoretical aspects of this proposal in detail. On the application side, specifically, we explore and analyze the performance of our CG algorithm with a specific focus on training deep models on the constrained formulation shown in (5.2). Progressively, we go from cases where there is no (or negligible) loss of both accuracy and training time to scenarios where this procedure shines and offers significant benefits in performance, runtime and generalization. Our experiments indicate that: (i) **with less than** 50% **#-parameters**, we improve ResNet accuracy by 25% (from 8% to 6% test error), and (ii) GANs can be trained in nearly a **third of the computational time** achieving the same or better qualitative performance on an image inpainting task.

5.3 First Order Methods: Two Representatives

To setup the stage for our development we will review the two broad strategies that are used to solve problems of the form shown in (5.2). **First**, a natural extension of gradient descent (GD) also known as Projected GD (PGD) may be used. Intuitively, we take a gradient step and then compute the point that is closest to the feasible set

defined by the regularization function. Hence, at each iteration PGD requires the solution of the following optimization problem or the so-called Projection operator,

$$W_{t+1}^{PGD} \leftarrow \arg\min_{W:R(W)\leqslant\lambda} \frac{1}{2}\|W-(W_t-\eta g_t)\|_F^2 \tag{5.3}$$

where $\|W\|_F$ is the Frobenius norm of W, g_t is (an estimate of) the gradient of $\mathcal{L}$ at W_t and η is the step size. In practice, we compute g_t by using only a few training samples (or minibatch) and running backpropagation. Note that the objective $\mathbb{E}_{(x,y)\sim\mathcal{D}}\mathcal{L}(W)$ can be assumed to be smooth in W if we convolve with any probability distribution $\mathcal{D}'$ (close to $\mathcal{D}$ but) with a density function and is commonly referred to as *stochastic smoothing*. Hence, for our descriptions, we will assume that the derivative is well defined. Note that stochastic smoothing does not preserve sparsity patterns, and hence cannot be used to construct coreset, as in Chapter 4. Furthermore, when there are no constraints, (5.3) is simply the standard SGD method that requires optimizing a quadratic function on the feasible set. So, the main bottleneck in explicitly imposing constraints with PGD is the complexity of solving (5.3). Even though many $R(\cdot)$ do admit an efficient procedure in theory, using them for applications in training deep models has been a challenge since they may be complicated or not easily amenable to a GPU implementation Taylor et al. (2016); Frerix et al. (2017).

So, a natural question to ask is whether there are methods that are faster in the following sense: can we solve simpler problems at each iteration and also explicitly impose the constraints effectively? As previously discussed in Chapter 2, an assertive answer is provided by a scheme that falls under the **second** general category of first order methods: the Conditional Gradient (CG) algorithm, also known as the Frank-Wolfe algorithm, as in Chapter 4. Recall that CG methods solve the following *linear minimization problem at each iteration* instead of a quadratic one

$$s_t \in \arg\min_{W} g_t^T W \text{ s.t. } R(W)\leqslant\lambda \tag{5.4}$$

and update $W_{t+1}^{CG} \leftarrow \eta W_t + (1-\eta)s_t$. While both PGD and CG guarantee convergence with mild conditions on η, it may be the case (as we will see shortly) that problems of the form (5.4) can be *much simpler* than the form in (5.3) and hence highly suitable for training deep learning models. An additional bonus is that CG algorithms also offer nice space complexity guarantees that are also attainable in practice, making it a very promising choice for explicitly *constrained* training of deep models.

Remark 5.1. *As we discussed in Chapter 2, in order for CG algorithm (5.4) to be well defined, we need the feasible set to be bounded whereas this is not required for PGD (5.3) since the objective function is quadratic.*

To that end, we will see how the CG algorithm behaves for the regularization constraints that are commonly used.

Remark 5.2. *Note that although the loss function $\mathbb{E}\mathcal{L}(W)$ is nonconvex, the constraints that we need for almost all applications are convex, hence, all our algorithms are guaranteed to converge by design Lacoste-Julien (2016); Reddi et al. (2016) to a stationary point.*

5.4 Categorizing "Generic" Constraints for CG

We now describe how a broad basket of "generic" constraints broadly used in literature, can be arranged into a hierarchy — ranging from cases where a CG scheme is perfect and expected to yield wide-ranging improvements to situations where the performance is only satisfactory and additional technical development is warranted. For example, the ℓ_1-norm is often used to induce sparsity. The nuclear norm (sum of singular values) is used to induce a low rank regularization, often for compression and/or speed-up reasons.

So, how do we know which constraints when imposed using CG are likely to work well? In order to analyze the qualitative nature of constraints suitable for CG algorithm, we categorize the constraints into three categories based on how the updates will, computationally, and learning-wise, compare to a SGD update.

5.4.1 Category 1 constraints are excellent

We categorize constraints as Category 1 if both the SGD and CG updates take a similar form algebraically. The reason we call this category "excellent" is because it is easy to transfer the empirical knowledge that we obtained in the unconstrained setting, specifically, learning and dropout rates to the regime where we want to explicitly impose these additional constraints. Here, we see that we get quantifiable improvements in terms of both computation and learning.

Two types of generic constraints fall into this category: **1.** the Frobenius norm and **2.** the Nuclear norm Ruder (2017). We will now see how we can solve (5.3) and (5.4) by comparing and contrasting them.

1. **Frobenius Norm.** When $\mathsf{R}(\cdot)$ is the Frobenius norm, it is easy to see that (5.3) corresponds to the following,

$$W_{t+1}^{\text{PGD}} = \begin{cases} W_t - \eta g_t & \text{if } \|W_t - \eta g_t\|_{\mathsf{F}} \leqslant \lambda \\ \lambda \cdot \frac{W_t - \eta g_t}{\|W_t - \eta g_t\|_{\mathsf{F}}} & \text{otherwise,} \end{cases}$$

and (5.4) corresponds to $s_t = -\lambda \frac{g_t}{\|g_t\|_{\mathsf{F}}}$ which implies that,

$$W_{t+1}^{\text{CG}} = W_t - (1-\eta)\left(W_t + \lambda \frac{g_t}{\|g_t\|_{\mathsf{F}}}\right). \tag{5.5}$$

It is easy to see that both the update rules essentially take the same amount of calculation which can be easily done while performing a backpropagation step. So, the actual change in any existing implementation will be minimal but CG will automatically offer an important advantage, notably **scale invariance**, which several recent papers have found to be advantageous Lacoste-Julien and Jaggi (2015).

2. **Nuclear norm.** On the other hand, when $\mathsf{R}(\cdot)$ is the nuclear norm, the situation where we use CG (versus not) is quite different. All known projection (or proximal) algorithms require computing at each iteration the **full singular value decomposition** of W, which in the case of deep learning methods becomes restrictive Recht et al. (2010). In contrast, CG only requires computing the top-1 singular vector of W which can be done easily and efficiently on a GPU via the power method Jaggi (2013). Hence in this case, if the number of edges in the network is $|\mathsf{E}|$ we get a *near-quadratic speed up*, i.e., from $\mathsf{O}(|\mathsf{E}|^3)$ for PGD to $\mathsf{O}(|\mathsf{E}| \log |\mathsf{E}|)$ making it practically implementable Golub and Van Loan (2012) in the very large scale settings Yu et al. (2017). Furthermore, it is interesting to observe that the rank of W after running T iterations of CG is at most T which implies that we need to only store $2\mathsf{T}$ vectors instead of the whole matrix W making it a viable solution for deployment on devices with memory constraints Howard et al. (2017). The main takeaway is that, since projections are computationally expensive, **projected SGD is not a viable option in practice.**

5.4.2 Category 2 constraints are potentially good

As we saw earlier, CG algorithms are always *at least* as efficient as the PGD updates: in general, any constraint that can be imposed using the PGD algorithm can also be imposed by CG algorithm, if not faster. Hence, generic constraints are defined to be Category 2 constraints for CG if the empirical knowledge cannot be easily transferred from PGD. Two classical norms that fall into this category: $\|W\|_1$ and $\|W\|_\infty$.

1. PGD on the ℓ_1 ball can be done in linear time (see Duchi et al. (2008)) and for $\|W\|_\infty$ using gradient clipping Boyd and Vandenberghe (2004a). So, let us evaluate the CG step (5.4) for the constraint $\|W\|_1 \leqslant \lambda$ which corresponds to,

$$s_t^j = \begin{cases} -\lambda & \text{if } j^* = \arg\max_j \left| g_t^j \right| \\ 0 & \text{otherwise.} \end{cases} \tag{5.6}$$

   That is, we assign $-\lambda$ to the coordinate of the gradient g_t that has the maximum magnitude in the gradient matrix which corresponds to a *deterministic* dropout regularization in which at each iteration we only update *one* edge of the network. While this might not be necessarily bad, it is now common knowledge that a high dropout rate (i.e., updating very few weights at each iteration) leads to underfitting or in other words, the network tends to need a longer training time Srivastava et al. (2014).

2. Similarly, the update step (5.4) for CG algorithm with $\|W\|_\infty$ takes the following form,

$$s_t^j = \begin{cases} +\lambda & \text{if } g_t^j < 0 \\ -\lambda & \text{otherwise.} \end{cases} \tag{5.7}$$

   In this case, the CG update uses only the sign of the gradient and does not use the magnitude at all. In both cases, one issue is that information about the gradients is not used by the standard form of the algorithm making it not so efficient for practical purposes.

Interestingly, even though the update rules in (5.6) and (5.7) use extreme ways of using the gradient information, we can, in fact, use a group norm type penalty to

model the trade-off. Recent work in (Garber and Hazan, 2015) shows that there are very efficient procedures to solve the corresponding CG updates as well (5.4).

Remark 5.3. *The main takeaway from the discussion is that Category 2 constraints surprisingly unifies many regularization techniques that are traditionally used in DL in a more methodical way.*

5.4.3 Category 3 constraints need more work

There is one class of regularization norms that do not nicely fall in either of the above categories, but is used in several problems in vision: the Total Variation (TV) norm. TV norm is widely used in denoising algorithms to promote smoothness of the estimated sharp image Chambolle and Lions (1997). The TV norm on an image I is defined as a certain type of norm of its discrete gradient field $(\nabla_i I(\cdot), \nabla_j I(\cdot))$ i.e.,

$$\|I\|_{TV}^p := ((\|\nabla_i I\|_p + \|\nabla_j I\|_p)^p .$$

Note that for $p \in \{1, 2\}$, this corresponds to the classical anisotropic and isotropic TV norm respectively. Motivated by the above idea, we can now define the TV norm of a Feed Forward Deep Network. TV norm, as the name suggests, captures the notion of balanced networks, shown to make the network more stable Neyshabur et al. (2015a). Let A be the incidence matrix of the network: the rows of A are indexed by the nodes and the columns are indexed by the (directed) edges such that each column contains exactly two nonzero entries: a $+1, -1$ in the rows corresponding to the starting node u and ending node v respectively. Let us also consider the weight matrix of the network as a vector (for simplicity) indexed in the same order as the columns of A. Then, the TV norm of the deep neural network is,

$$\|W\|_{TV} := \|AW\|_p. \tag{5.8}$$

It turns out that when $R(W) = \|W\|_{TV}$, PGD is **not** trivial to solve and requires special schemes Fadili and Peyré (2011) with runtime complexity of $O(n^4)$ where n is the number of nodes — impractical for most deep learning applications. In contrast, CG iterations only require a special form of maximum flow computation which can be done efficiently Harchaoui et al. (2014).

Lemma 5.4. *An ϵ-approximate CG step (5.4) can be computed in $O(1/\epsilon)$ time (independent of dimensions of A).*

Proof. Harchaoui et al. (2014) showed that in order to solve the CG step for TV norm we first solve

$$\max_{\beta \in \mathbb{R}, f} \beta \text{ s.t. } Af = \beta g_t, f \in [0,1]^E, \tag{5.9}$$

where E is the number of edges of the network and then compute the Lagrange multipliers at the optimal solution of problem (5.9). Instead of first solving (5.9), we propose to solve the dual directly using continuous optimization techniques. Introducing dual variables v for the equality constraints and w for the box constraints, our dual linear program can be written as,

$$\max_{w,v} 1^T w \quad \text{s.t.} \quad -A^T v + w \leqslant 0,\ g_t^T v = -1,\ w \leqslant 0. \tag{5.10}$$

Since v is a free variable, problem (5.10) is equivalent to,

$$\max_{w,v} 1^T w \quad \text{s.t.} \quad w \leqslant A^T v,\ g_t^T v = 1,\ w \leqslant 0.$$

Eliminating w, we can write the dual of (5.9) as,

$$\min_{v \in \mathbb{R}^E} \sum_{i=1}^{E} \min\left(0, v^T A_{[i,:]}\right) \text{ s.t. } g_t^T v = 1$$

where $A_{[i,:]}$ denotes the i-th row of A. Now it is easy to see that many continuous optimization methods are amenable which does *not* require us to instantiate the matrix A. Hence we can now use Theorem 1 in Johnson and Zhang (2013) gives us the desired result. □

Remark 5.5. *The above discussion suggests that conceptually, Category 3 constraints can be incorporated and will immensely benefit from CG methods. However, unlike Category 1-2 constraints, it requires specialized implementations to solve subproblems from (5.8) which are not currently available in popular libraries. So, additional work is needed before broad utilization may be possible.*

Algorithm 7 Path-CG iterations

Pick a starting point $W_0 : \|W_0\|_\pi \leqslant \lambda$ and $\eta \in (0,1)$.
for $t = 0, 1, 2, \cdots,$ T iterations **do**
  for $j = 0, 1, 2, \cdots,$ l layers **do**
    $g \leftarrow$ gradient of edges from $j-1$ to j layer.
    Compute $\gamma_e \; \forall \; e$ from $j-1$ to j layer (eq (5.11))
    Set $s_t^j \leftarrow \arg\min_W g^T W$ s.t. $\|\Gamma W\|_2 \leqslant \lambda$ (eq(5.12))
    Update $W_{t+1}^j \leftarrow \eta W_t^j + (1-\eta) s_t^j$
  end for
end for

5.5 Path Norm Constraints in Deep Learning

So far, we only covered constraints that were already in use in vision/machine learning and recently, some attempts Márquez-Neila et al. (2017) were made to utilize them in deep networks. Now, we review a new notion of regularization, introduced recently, that has its roots primarily in deep learning Neyshabur et al. (2015a). We will first see the definition and explain some of the properties that this type of constraint captures.

Definition 5.6. *Neyshabur et al. (2015a) The ℓ_2-path regularizer is defined as :*

$$\|W\|_\pi^2 = \sum_{v_{in}[i] \xrightarrow{e_1} v_1 \xrightarrow{e_2} \cdots v_{out}[j]} \left| \prod_{k=1}^{l} W_{e_k} \right|^2 .$$

Here π denotes the set of paths, v_{in} corresponds to a node in the input layer, e_i corresponds to an edge between a node $(i-1)$-th layer and i-th layer that lies in the path between v_{in} and v_{out} in the output layer. Therefore, the path norm measures *norm of all possible paths π in the network up to the output layer*.

Why do we need path norm? One of the basic properties of ReLu (Rectified Linear Units) is that it is *scaling invariant* in the following way: multiplying the weights of incoming edges to a node i by a positive constant and dividing the outgoing edges from the same node i does not change $\mathcal{L}$ for any (x, y). Hence, an update scheme that is *scaling invariant* will significantly increase the training speed. Furthermore, the authors in Neyshabur et al. (2015a) showed how path regularization converges to optimal solutions that can generalize better compared to the usual SGD updates — so apart from computational benefits, there are clear statistical generalization advantages too.

How do we incorporate the path norm constraint? Recall from Remark 5.1 that the feasible set has to be bounded, so that the step (5.4) is well defined. Unfortunately, this is not the case with the path norm as shown in the following example.

Example 2. *Consider a simple line graph with scalar weights W_1 and W_2. In this case, there is only one path and the path norm constraint is $W_1^2 W_2^2 \leqslant 1$ which is clearly unbounded.*

Further, we are not aware of an efficient procedure to compute the projection for higher dimensions since there is no known efficient separation oracle. Interestingly, we take advantage of the fact that if we fix W_1, then the feasible set is bounded. This intuition can be generalized, that is, we can update one layer at a time which we will describe now precisely.

Path-CG Algorithm. In order to simplify the presentation, we will assume that there are no biases noting that the procedure can be easily extended to the case when we have individual bias for every node. Let us fix a layer j and the vectorized weight matrix of that layer be W that we want to update and as usual, g corresponds to the gradient. Let the number of nodes in the $(j-1)$ and j-th layers be n_1 and n_2 respectively. For each edge between these two layers we will compute the scaling factors γ_e defined as,

$$\gamma_e = \sum_{v_{in}[i] \cdots \xrightarrow{e} \cdots v_{out}[j]} \left| \prod_{e_k \neq e} W_{e_k} \right|^2 . \tag{5.11}$$

Intuitively, γ_e computes the norm of all paths that pass through the edge e excluding the weight of e. This can be efficiently done using Dynamic Programming in time $O(l)$ where l is the number of layers. Consequently, the computation of path norm also satisfies the same runtime, see Neyshabur et al. (2015a) for more details. Now, observe that the path norm constraint when all of the other layers are fixed reduces to solving the following problem,

$$\min_W g^T W \text{ s.t. } \|\Gamma W\|_2 \leqslant \lambda \tag{5.12}$$

where Γ is a diagonal matrix with $\Gamma_{e,e} = \gamma_e$, see (5.11). Hence, we can see that the problem again reduces to a simple rescaling and then normalization as seen for the Frobenius norm in (5.5) and repeat for each layer, see Figure 5.1.

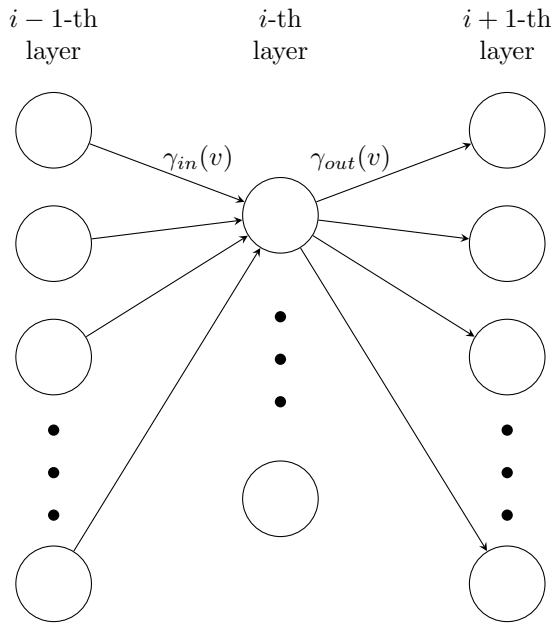


Figure 5.1: In order to compute the path norm, we can compute γ_{in} and γ_{out} for each node in the network. We then set $\gamma(u \to v) = \gamma_{in}(u)\gamma_{out}(v)$

Define $\gamma_{in}(v)$ and $\gamma_{out}(v)$ as,

$$\gamma_{in}(v) = \sum_{(u\to v)\in E} \gamma_{in}(u) w_{u,v}^2$$
$$\gamma_{out}(v) = \sum_{(v\to u)\in E} \gamma_{out}(u) w_{u,v}^2$$

Fix layer i that we want to update and let the g_i^t be the gradient of the loss function with respect to the weight matrix of the edges between $i-1$ and i layers. Then the squared path norm can be computed as the sum of squares (taken over all edges) of product of $\gamma_{uv} := \gamma_{in}(u)\gamma_{out}(v)$ and w_{uv}^2 for every edge in between the layers, that is,

$$\|W\|_\pi^2 := \sum_{uv} \gamma_{uv} w_{uv}^2 = \|\hat{\Gamma} W\|_F^2$$

where the matrix $\hat{\Gamma}$ is diagonal with elements being $\gamma_{uv}^{1/2}$. Hence we have that the

Path-CG subproblems correspond to solving

$$\min_{W} g^{\top} W \quad \text{s.t.} \quad \|\hat{\Gamma} W\|_2 \leqslant \lambda.$$

Remark 5.7. *The starting point W_0 such that $\|W_0\|_\pi \leqslant \lambda$ can be chosen simply by randomly assigning the weights from the Normal Distribution with mean 0.*

Complexity of Path-CG algorithm in Algorithm 7. From the above discussion, our full algorithm is given in Algorithm 7. The main computational complexity in Path-CG comes from computing the matrix Γ for each layer, but as we described earlier, this can be done *by* backpropagation – $O(1)$ flops per example. Hence, the complexity of Path-CG for running T iterations is essentially $O(lBT)$ where B is a size of the mini-batch.

Scale invariance of Path-CG algorithm in Algorithm 7. Note that CG algorithms satisfy a much general property called as Affine Invariance Jaggi (2013), which implies that it is also scale invariant. Scale invariance makes our algorithm more efficient (in wall clock time) since it avoids exploring parameters that correspond to the same prediction function.

5.6 Experiments

We present experimental results on three different case studies to support our basic premise and theoretical findings in the earlier sections: constraints can be easily handled with our CG algorithm in the context of Deep Learning while preserving the empirical performance of the models.

1. **The first set** of experiments is designed to show how simple/generic constraints can be easily incorporated in existing deep learning models to get both faster training times and better accuracy while reducing the #-layers using the ResNet architecture.

2. **The second set** of experiments is to evaluate our Path-CG algorithm. The goal is to show that Path-CG is much more stable than the Path-SGD algorithm in Neyshabur et al. (2015a), implying lower generalization error of the model.

3. **In the third set** of experiments we show that GANs (Generative Adversarial Networks) can be trained faster using the CG algorithm and that the training

tends to be stable. To validate this, we test the performance of the GAN on image inpainting.

Since CG algorithm maintains a solution that is a convex combination of all previous iterates, hence to decrease the effect of random initialization, the training scheme consists of two phases:

1. **Burn-in phase** in which the CG algorithm is run with a constant stepsize.
2. **Decay phase** in which the stepsize is decaying according to $1/t$. This makes sure that the effect of randomness from the initialization is diminished.

We use 1 epoch for the burn-in phase, hence we can conclude that the algorithm is guaranteed to converge to a stationary point Lacoste-Julien (2016).

5.6.1 Improve ResNets using Conditional Gradients

We start with the problem of image classification, detection and localization. For these tasks, one of the best performing architectures are variants of the Deep Residual Networks (ResNet) He et al. (2016). For our purposes, to analyze the performance of CG algorithm, we used the shallower variant of ResNet, namely ResNet-32 (32 hidden layers) architecture and trained on the CIFAR10 Krizhevsky et al. (2012) dataset. ResNet-32 consists of 5 residual blocks and 2 fully connected, one each at the input and output layers. Each residual block consists of 2 convolution, ReLu (Rectified Linear units), and batch normalization layers, see He et al. (2016) for more details. CIFAR10 dataset contains 60000 color images of size 32×32 with 10 different categories/labels. Hence, the network contains approximately 0.46M parameters.

To make the discussion clear, we present results for the case where the total Frobenius norm of the network parameters is constrained to be less than λ and trained using the CG algorithm. To see the effect of the parameters λ and step sizes η on the model, we ran 80000 iterations, see Figure 5.2. The plots essentially show that if λ is chosen reasonably big, then the accuracy of CG is very close to the accuracy of ResNet-164 (5.46% top-1 test error, see He et al. (2016)) that has **many more parameters** (approximately 5 times!). In practice, since λ is a constraint parameter, we can initially choose λ to be small and gradually increase it, thus avoiding complicated grid search procedures.Thus, Figure 5.2 shows that CG can be used to improve the performance of *existing architectures* by appropriately choosing constraints.

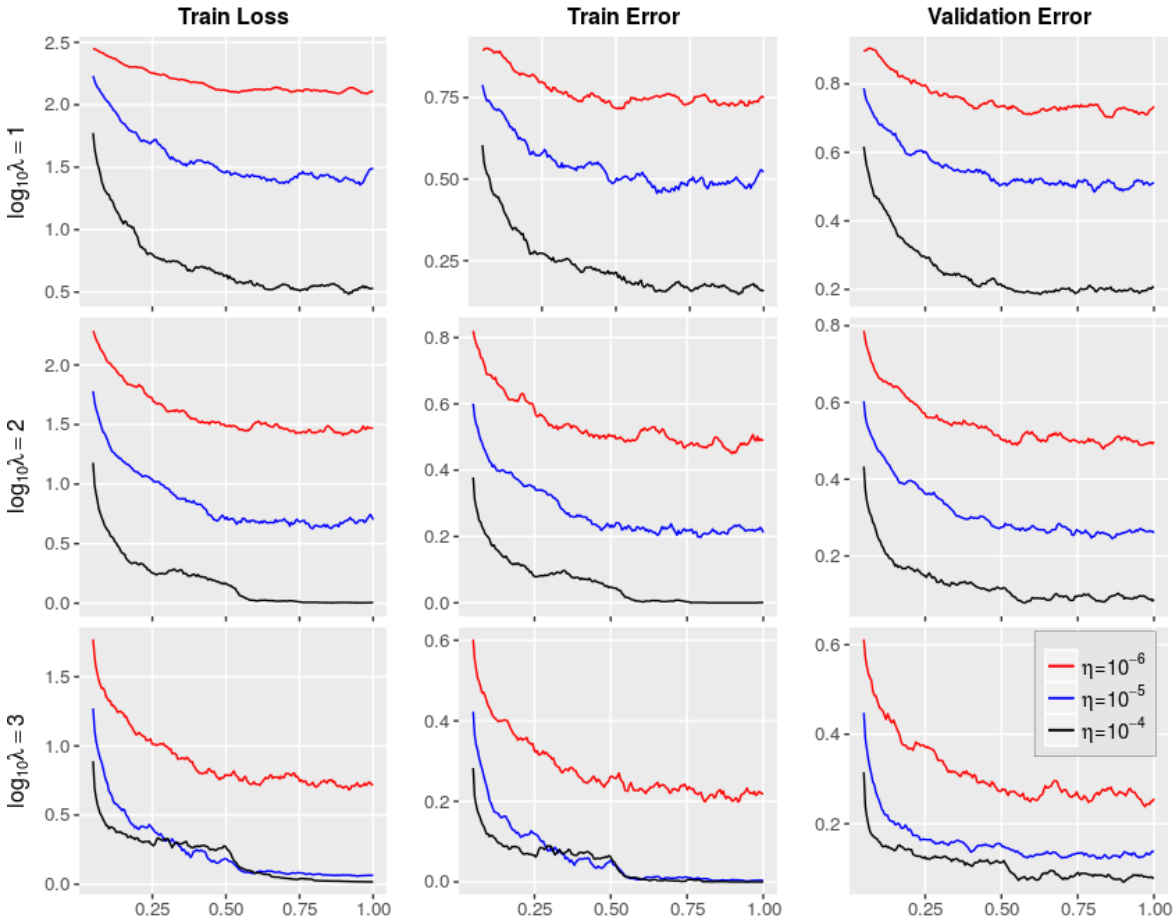


Figure 5.2: Performance of CG on ResNet-32 on CIFAR10 dataset (x-axis denotes the fraction of T): as λ increases, the training error, loss value and test error all start to decrease simultaneously.

Takeaway. *CG offers fewer parameters and higher accuracy on a standard network with no additional change.*

5.6.2 Path-CG vs Path-SGD: Which is better?

In this case study, the goal is to compare Path-CG with the Path-SGD algorithm Neyshabur et al. (2015a) in terms of both accuracy and stability of the algorithm. To that end, we considered image classification problem with a path norm constraint on the network: $\|W_t\|_p \leqslant \lambda$ for varying λ as before. We train a simple feed-forward network which consists of 2 fully-connected hidden layers with 4000 units each, followed by the output layer with 10 nodes. We used ReLu nonlinearity as the

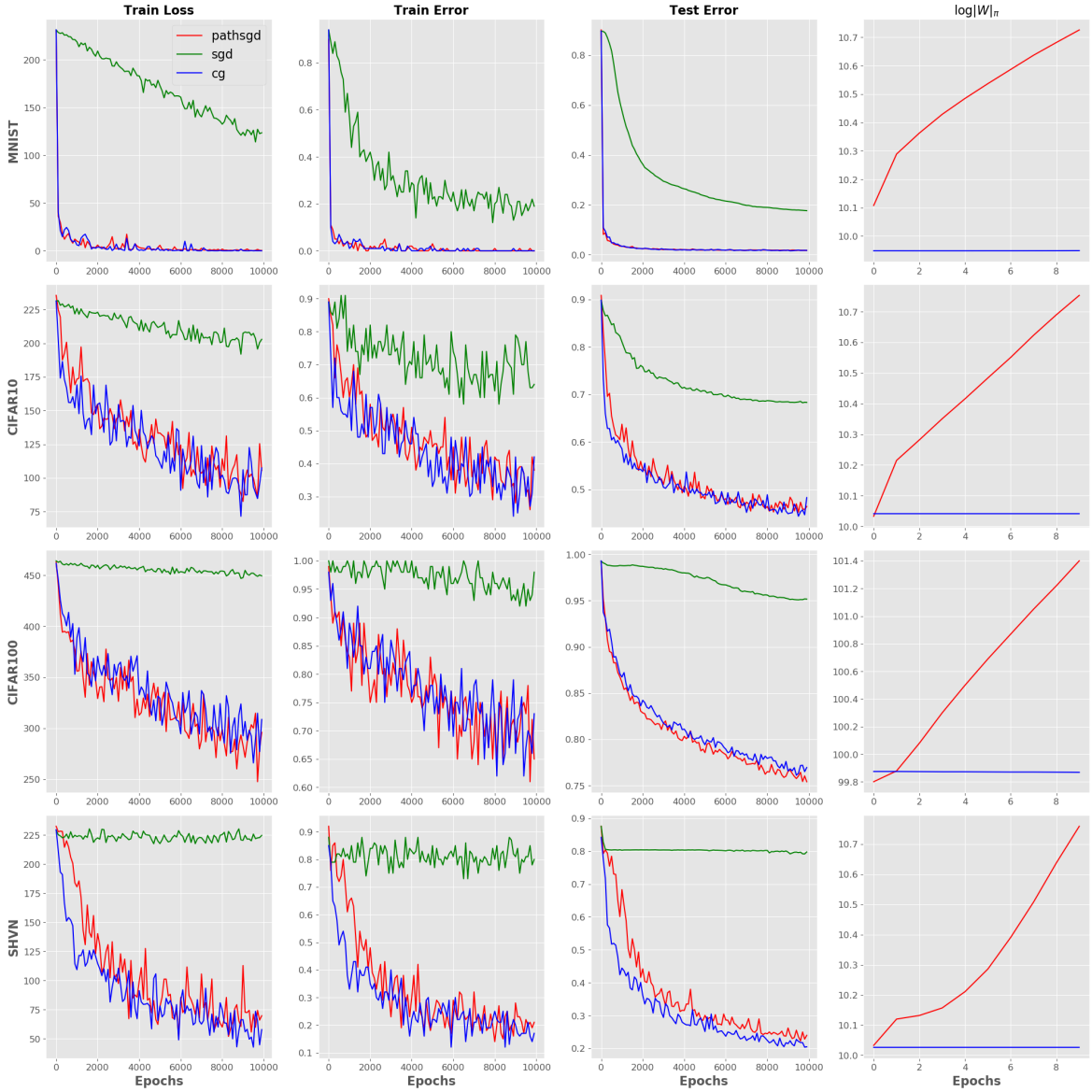


Figure 5.3: Performance of Path-CG vs SGD on a 2-layer fully connected network on four datasets (x-axis denotes the #iterations). Observe that across all datasets, Path-CG is much faster than SGD (first three columns). Last column shows that SGD is not stable with respect to the path norm.

activation function and cross entropy as the loss defined by,

Definition 5.8 (Cross entropy loss)**.** *Fix ground truth* y *such that* $y \in \Delta$*, where* $\Delta \subset \mathbb{R}^M$ *is the probability simplex as in Section 1.4. Then, the cross entropy loss at an arbitrary*

distribution $z \in \Delta$ *is given by,*

$$CE_y(z) := -\sum_{c=1}^{M} y_c \log z_c.$$

We performed experiments on 4 standard datasets for image classification: MNIST, CIFAR (10,100) Krizhevsky et al. (2012) and finally color images of house numbers from SVHN dataset Netzer et al.. Figure 5.3 shows the result for $\lambda = 10^{-5}$ (after tuning), it can achieve the same accuracy as that of Path-SGD.

Path-CG has one main advantage over Path-SGD: Our results show that Path-CG is more stable while the path norm of Path-SGD algorithm increases rapidly. This shows that Path-SGD *does not* effectively regularize the path norm whereas Path-CG keeps the path norm less than λ as expected.

Takeaway. *All statistical benefits of path norm are possible via CG while being computationally stable.*

5.6.3 Image Inpainting using Conditional Gradients

Finally, we illustrate the ability of our CG framework on an exciting and recent application of image inpainting using Generative Adversarial Networks (GANs). We now briefly explain the overall experimental setup. GANs using game theoretic notions can be defined as a system of 2 neural networks called Generator and the Discriminator competing with each other in a zero-sum game Arora et al. (2017).

Image inpainting/completion can be performed using the following two steps Amos (2016):

1. Train a standard GAN as a normal image generation task, and

2. Use the trained generator and then tune the noise that gives the best output.

Hence, our hypothesis is that if the generator is trained well, then the follow-up task of image inpainting benefits automatically.

Train DC-GAN faster for better image inpainting. We used the state of the art DC-GAN architecture in our experiments and we impose a Frobenius norm constraint on the parameters but *only* on the Discriminator to avoid mode collapse issues and trained using the CG algorithm. Figure 5.4 illustrates our overall procedure in which a GAN is used to learn the training distribution of images, and then used for the

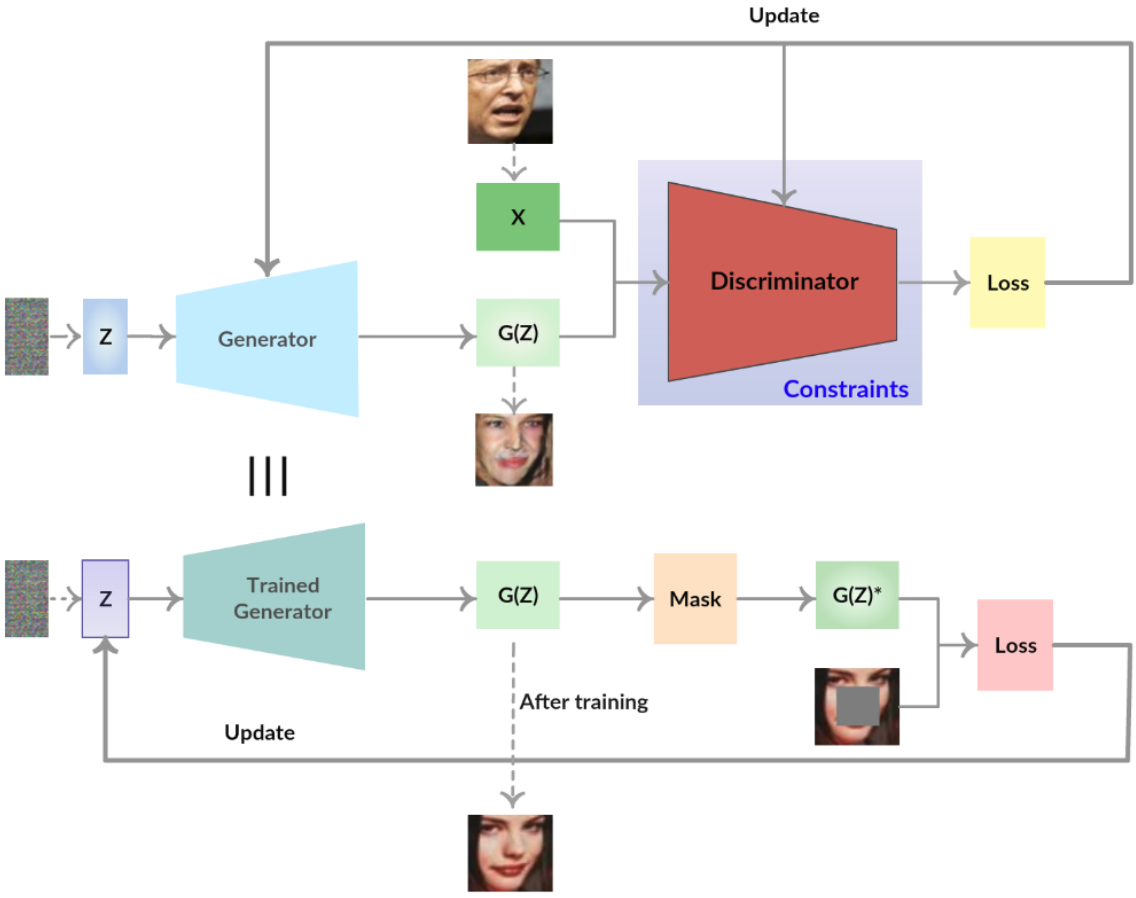


Figure 5.4: Illustrates the task of image inpainting overall pipeline.

inpainting task. Qualitative results are shown in Figure 5.5. In order to verify the performance of the CG algorithm, we used 2 standard face image datasets from CelebA and LWF and conducted two experiments: trained on the CelebA dataset with LFW being the test dataset and vice-versa. We found that the generator generates very high quality images after being trained with LFW images in comparison to the original DC-GAN *in just* 10 *epochs* (reducing the computational **cost by** 50%). Quantitatively, we provide numerical evidence in Figure 5.6 with 2 intrinsic metrics viz., **Structural Similarity (SSim), Mean Squared Error (MSE)** and 1 extrinsic metric, **Frechet Inception Distance (FID)**. All the three metrics are standard in GAN literature. We calculated the intrinsic metrics *after* the image completion phase. We can see that on *all the three metrics*, CG outperforms SGD clearly.

Takeaway. *GANs can be trained faster with no change in accuracy.*

Figure 5.5: CG-trained DC-GAN performs as good as (or better than) SGD-based DC-GAN but with 50% epochs. From Left to Right: Column 1 represents an input masked image, Column 2 represents the Ground Truth, Column 3 represents the output of the GAN trained using CG Algorithm, Column 4 represents the output of the GAN trained using SGD Algorithm.

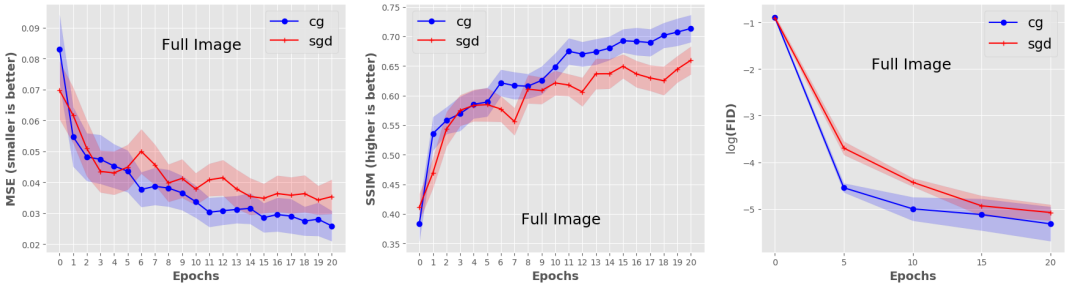


Figure 5.6: From left: show MSE/SSIM/FID on the **full** image.

5.7 Conclusions

The main emphasis of our work is to provide evidence supporting three distinct but related threads:

1. Global constraints are relevant in the context of training deep models in vision and machine learning.

2. The lack of support for global constraints in existing libraries like Keras and Tensorflow Abadi et al. (2016) may be because of the complex interplay between constraints and SGD which we have shown can be side-stepped, to a great extent, using CG.

3. Constraints can be easily incorporated with negligible to small changes to existing implementations.

We provide empirical results on *three* different case studies to support our claims, and conjecture that a broad variety of other problems will immediately benefit by viewing them through the lens of CG algorithms. Our analysis and experiments suggest concrete ways in which one may realize improvements, in both generalization and runtime, by substituting in CG schemes in deep learning models. Tensorflow code for all our experiments is available in Github here.

Chapter 6

Experimental Design for Sparse Models

In this chapter, we explore a discrete optimization problem in which constraints arise due to practical considerations involved in performing a longitudinal observational study, for example, based on longitudinal acquisitions of neuroimaging and cognitive data over many years. Interestingly, we find that continuous relaxations of nonlinear discrete problems arising in this setting can be solved efficiently in practice. A key benefit of our formulation is that the *rounding* a fractional solution can be done without much loss of accuracy.

6.1 Introduction

Experimental Design (ED) is a problem with deep foundations dating back at least to the early 1900s Kempthorne (1952); Kirk (1982). Here, given the covariates x_i's, an *experimenter* must conduct an *experiment* in order to obtain the value of the dependent (or response) variables y_i's. The focus of much of the classical work on this topic is to maximize the amount of information that the full experiment yields for a given (or least) amount of work. In many situations, each experiment or measurement may have a monetary cost associated with it. Given a fixed budget $B \in \mathbb{R}_+$, the budgeted version of the experimental design problem is to choose the set of x_i's for which we will conduct an experiment to obtain the corresponding y_i's, while satisfying the budget constraint. As a simple example, the budget constraint may restrict the number of y_i's we may request, i.e., the experimenter can perform at most

B experiments. As before, the selected subset must be a good prototype for the entire dataset for model estimation purposes, e.g., $\beta : x_i \rightarrow y_i$. These problems are studied under the umbrella of optimal design Pukelsheim (1993).

In recent years, there is a renewed interest in ED since it provides a framework to study numerous adaptive data acquisition scenarios. While randomization offers one solution Niu et al. (2011), recent results demonstrate that optimization based schemes yield a competitive alternative Bertsimas et al. (2015). Solutions to a number of interesting variants of the problem have been proposed, for instance, Horel et al. (2014) provides approximation guarantees for the convex relaxation of the problem. In vision, the availability of crowd-sourced platforms has led to scenarios where we seek to acquire low cost reliable data; trusted workers charge a premium per HIT Li and Guo (2013).

Apart from these results, active learning approaches also make use of ED concepts, but the selection process there is sequential. An important distinction in active learning is that the algorithm chooses the next (subset of) examples which the back-end machine learning algorithm needs labeled. The algorithm then proceeds and request labels for more examples iteratively. Most ED formulations do *not* offer this flexibility. Specifically, while these methods may try to minimize the number of queries or labelings required (for a pre-determined accuracy), we study a related but distinct question. If we fix the number of queries a priori, we study the issue of choosing a subset such that the corresponding estimator is close to the estimator inferred from the full dataset (i.e., y_i for *all* x_i were available). Further, the algorithm gets *no more chances to query the experimenter*. For this formulation to make sense, the criteria for subset selection must be closely tied to the later statistical estimation task that the subset of samples will be used for. To our knowledge, existing solutions to the general version of this problem do not scale up to large datasets (n large), see Dette et al. (2011). Moreover, often we need to use sampling schemes to approximate an integral at each step which is a *nontrivial* for large p due to the *high dimensionality* Konstantinou et al. (2011).

Application. One motivating application is conducting cost effective imaging-based neuroscience studies; this setting will be used to evaluate our proposed models. In Alzheimer's Disease (AD), the problem of *predicting* future cognitive decline is important for a broad variety of reasons Landau et al. (2010); Hinrichs et al. (2011). Identifying decliners is of direct relevance in disease progression studies and also

serves the goal of maximizing statistical power in trials Ithapu et al. (2015) – both of which are based on longitudinal changes in participants Searcey et al. (1994); Hinrichs et al. (2012). The dependent variable of interest here is change in cognition and/or diagnostic scores, over time. The independent variables include imaging (and imaging-derived) measures, genetic and other data acquired at the *baseline* time-point (or initial visit). Here, keeping a subject enrolled in the study (e.g., for a second visit) is expensive, but will provide the response y_i for subject i. The goal is to choose a "subset" of all subjects that can help estimate the parameters of the model, without affecting the statistical power.

Our Results. In Section 6.3, we propose two formulations for the problem. The first model is motivated geometrically while the second one involves certain algebraic manipulations. Experimentally we show that both models yield consistent results, with each other as well as with the "full" model. In Section 6.4, we evaluate our algorithms on a large neuroimaging dataset ($\approx$ 1000 subjects) using both qualitative and quantitative performance measures. Empirical results show that our algorithms are robust and promising for formulations involving sparse linear models.

6.2 Preliminaries

Consider the linear model $y_i = x_i^T\beta + \epsilon$ where $x_i, \beta \in \mathbb{R}^p, y_i \in \mathbb{R}$ and $\epsilon \sim \mathcal{N}(0,1)$. The regression task for β is,

$$\beta^* := \arg\min_{\beta} \frac{1}{2}\|X\beta - y\|_u^v + \epsilon g(\beta)$$

where the rows of $X \in \mathbb{R}^{n\times p}$ correspond to samples (or data instances, subjects). Here, g is a penalty function that specifies desired properties or characteristics of the optimal regressor β^* and ϵ is the Tikhonov regularization parameter. We assume that $u = v = 2$ unless otherwise specified which corresponds to the standard linear regression loss function. Recall that when $g(\beta) = \beta^T M\beta$ for some $M \succ 0$, then β^* has a closed form solution $\beta^* = \left(X^TX + \epsilon M\right)^{-1} X^Ty$ also known as *ridge* regression. Ridge regression is particularly useful when $p > n$ because X^TX is singular or when the covariates are highly correlated, where a typical regressor may overfit rather than explaining the underlying process. So, the ability to adjust the regularizer enables the estimation process. There are some obvious choices for M, e.g., $M = I$ corresponds

to the least norm least squares solution. On the other hand, when $p < n$ and if the rank of X is p, Li (2008) shows that the ridge regression is robust to noise or outliers.

6.2.1 Related Work

The literature analyzing experimental design procedures for the closely related problem of ridge regression is quite extensive. The ED problem for ridge regression can be written as the following (combinatorial) problem,

$$S^* := \arg\max_{|S| \leqslant B} f\left(\sum_{i \in S} x_i x_i^T + \epsilon I\right)$$

where we identify S with the set of selected subjects for a budget B. This problem can be equivalently formulated as,

$$S^* = \arg\max_{\mu \in \{0,1\}^n} f\left(\sum_{i=1}^{n} \mu_i x_i x_i^T + \epsilon I\right) \quad \text{s.t.} \quad \mathbf{1}^T \mu \leqslant B, \tag{6.1}$$

where $\mathbf{1} \in \mathbb{R}^n$ is the vector of all 1s. The choice of f determines the nature of the regressor from the selected subset, for example, $f(\cdot) = \log\det(\cdot)$ is referred to as the D−optimality criterion. Intuitively, a D−optimal design corresponds to the set of subjects that maximizes the information gain. There are other choices for the objective, see Das; Pukelsheim (1993); Chaloner and Verdinelli (1995). A common feature of many optimality criteria is that they lead to convex problems when the integrality constraints are relaxed. There are efficient ways to solve this relaxed problem — in particular, when Frank Wolfe type methods are employed, the number of nonzero entries of μ has a relationship with the number of iterations Jaggi (2013). Once the relaxed problem is solved, pipage rounding schemes yield a solution without sacrificing the objective function value much Ageev and Sviridenko (2004).

The case for ℓ_1. While ridge regression has many attractive properties, the solutions from ridge regression may have many nonzero entries that are close to zero Tibshirani (1996). Recent results suggest that in various cases it may be more appropriate to use the ℓ_1-norm instead – it induces sparsity in the optimal regressor and hence the model or the regressor may be more interpretable Candes and Tao (2005); Candès and Plan (2009). When the ℓ_1-regularization is used, coordinates with nonzero entries in the optimal regressor correspond to features that are responsible in

the "linearity" of the model, and explains the "selection" aspect of LASSO. After this procedure, a ridge regression problem is solved only on this reduced set of features that were selected by LASSO as described in Tibshirani (1996). The problem of interest is $\beta_1^* \in \arg\min_\beta \frac{1}{2}||X\beta - y||_2^2 + \epsilon||\beta||_1$. Under some mild conditions, we can assume that β_1^* is unique and so replace the containment operator with equality. Unlike ridge regression, iterative procedures are needed here since β_1^* does not have a closed form expression.

6.3 Our proposed formulations

Some basic assumptions. We first clarify a few basic assumptions to simplify the presentation. If necessary, we will add ϵI to the covariance matrix so that the corresponding inverse and the $\log\det(\cdot)$ operations are meaningful. We also assume without loss of generality that $||x_i||_2 \leqslant 1$, in other words, X is divided by the maximum sample norm. The constraint $\mathbf{1}^T\mu \leqslant B$ can be replaced by a more general $d^T\mu \leqslant B$ for $d > 0$, i.e., where the cost of selecting different subjects is different. Finally, we fix $f(\cdot)$ to be $\log\det(\cdot)$ (corresponding to the D−optimality criterion) since it is conceptually simpler. Our algorithms remain unchanged if f is replaced by another smooth convex function.

We now describe the two models for the ED problem (for LASSO): the first formulation in Sec. 6.3.1 is motivated geometrically whereas the next one in Sec. 6.3.2 involves certain algebraic manipulations but offers some efficiency benefits.

6.3.1 ED-S: Spectral Experimental Design

The ED-S approach is driven by a simple geometric interpretation of the LASSO. Consider the following two equivalent formulations of ridge regression and LASSO,

$$\begin{aligned}
\beta^* &= \arg\min_\beta \frac{1}{2}||X\beta - y||_2^2 + \lambda||\beta||_2^2 \quad \text{(RIDGE)} \\
&\equiv \arg\min_\beta \frac{1}{2}||X\beta - y||_2^2 \ \text{ s.t. } \ ||\beta||_2^2 \leqslant \tau \\
\beta_1^* &= \arg\min_\beta \frac{1}{2}||X\beta - y||_2^2 + \lambda||\beta||_1 \quad \text{(LASSO)} \\
&\equiv \arg\min_\beta \frac{1}{2}||X\beta - y||_2^2 \ \text{ s.t. } \ ||\beta||_1 \leqslant \tau_1
\end{aligned} \tag{6.2}$$

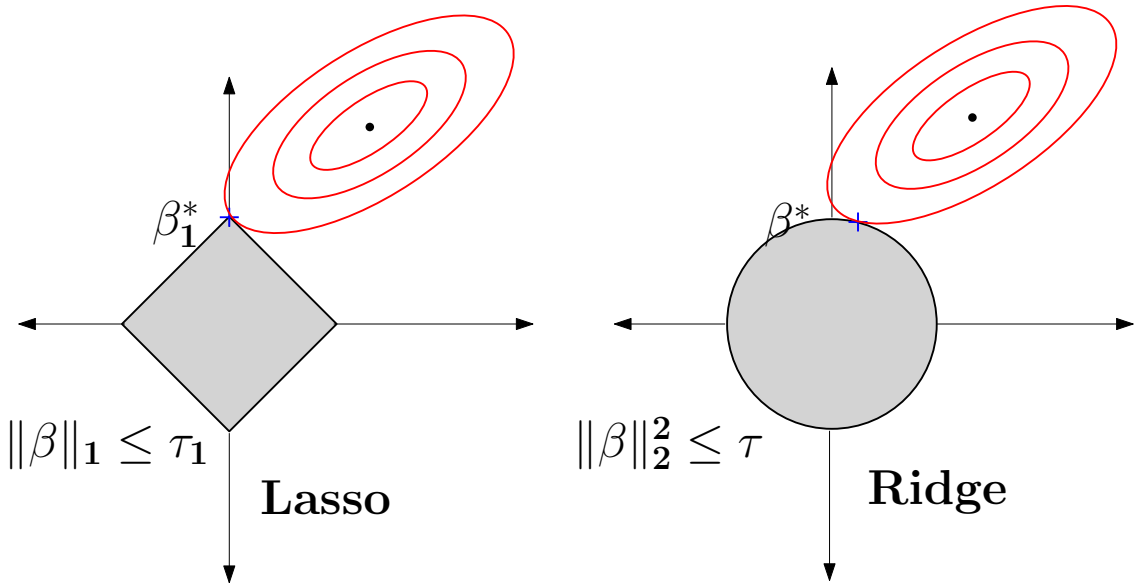


Figure 6.1: Variable selection of Lasso estimate β_1^*

for positive scalars τ and τ_1. The optimal solution in both the cases is where the objective function (identical for both problems) touches the feasible set: the ℓ_1 (and ℓ_2 norm) balls respectively. The difference between the problems is that the $\{\beta : \|\beta\|_1 \leqslant \tau_1\}$ is polyhedral (compact) and so has a finite number of extreme points given by $\{\pm e_i\}$, where e_i is the standard basis vectors in appropriate dimensions, see Fig. 6.3.1. In (6.2), the objective function is likely to touch a vertex of the ℓ_1 ball, and so yields sparse solutions.

6.3.1.1 A Curvature Matching Approach

Our proposal is motivated by the following simple observation: consider the full setting where we have access to the entire set of y_i's and the reduced setup where y_i's are available only for a subset. Intuitively, if the objective function of the full and the reduced setups behave similarly, then the corresponding regression coefficients will also be similar. To obtain this desired property, we may ask that the *reduced objective function to have approximately the same curvature as the full one*. Recall that the Hessian carries most of the curvature information and the optimal value. That is, eigenvalues of the Hessian are called "principal curvatures" in differential geometry and play a critical role in analyzing first order methods Kingma and Ba (2014). So, ED-S may

offer strong guarantees directly if, the spectrum of full and reduced sets are the same; then, the iterates generated (and optimal solutions) will be similar. To that end, we require that the eigen vectors of the Hessian to be close to each other which may be accomplished by making sure that the geometry of the reduced setup is preserved relative to the full one. The unknowns, μ, will correspond to the selection of samples for the reduced setup. Succinctly, the ED problem can be formulated as ($\lambda \geqslant 0$),

$$\begin{aligned} &\max_{\mu \in \{0,1\}^n} \log\det\left(\sum_{i=1}^{n} \mu_i x_i x_i^T + \epsilon I\right) + \lambda R_\Gamma(\Gamma_\mu) \\ &\text{s.t. } \mathbf{1}^T\mu \leqslant B \end{aligned}$$

where Γ contains the eigen vectors of the entire X^TX and Γ_μ represents the eigen vectors of the chosen subjects given by μ. $R_\Gamma(\cdot)$ is a (smooth) concave function that encourages similarity between the eigenvectors of the full Hessian and the reduced Hessian. While the log det term captures the linearity (D−optimal) in the regression problem, R_Γ captures the geometry. That is, the log det term corresponds to D-optimality for linear regression as mentioned earlier.

We can now proceed to write the explicit formulation of the problem. For simplicity, we only promote similarity between the top eigen vector between the Hessians noting that the top k−eigen vectors case ($k \leqslant p$) is easy as well. Let γ be the eigenvector corresponding to the largest eigenvalue and u be the decision variable for the largest eigenvector of the reduced Hessian. With this notation, taking $R(\cdot)$ to be the squared loss and $(\lambda \geqslant 0)$, we seek to solve,

$$\begin{aligned} &\min_{\mu,u} \log\det\left(\sum_{i=1}^{n} \mu_i x_i x_i^T + \epsilon I\right)^{-1} + \lambda\|\gamma - u\|_2^2 \qquad (6.3) \\ &\text{s.t. } 0 \leqslant \mu \leqslant 1, \mathbf{1}^T\mu \leqslant B, \\ &\quad u \in \arg\max_{v} \left\{ v^T \left(\sum_{i=1}^{n} \mu_i x_i x_i^T + \epsilon I\right) v \text{ s.t. } v^Tv = 1 \right\} \end{aligned}$$

6.3.1.2 Algorithm

We will now use some simple manipulations to obtain an equivalent formulation of the above problem for which efficient algorithms can be designed.

The largest eigenvector of a symmetric positive definite matrix can be written as an optimization problem,

$$\arg \max_{u:||u||_2^2=1} u^T M u = \arg \min_{u:||u||_2^2=1} u^T M^{-1} u$$

for a given symmetric positive definite matrix M. Therefore, our formulation can be written as,

$$\min_{\mu,u} \log\det\left(\sum_{i=1}^{n} \mu_i x_i x_i^T + \epsilon I\right)^{-1} + \lambda||\gamma - u||_2^2 + u^T\left(\sum_{i=1}^{n} \mu_i x_i x_i^T + \epsilon I\right)^{-1} u \quad (6.4)$$
$$\text{s.t. } 0 \leqslant \mu \leqslant 1, \quad \mathbf{1}^T\mu \leqslant B, \quad ||u||_2^2 = 1$$

The above problem is nonconvex because of the squared norm constraint. But we note two important aspects of (6.4). First, if we fix μ, we obtain a subproblem with a convex objective with one orthogonality constraint, we will call this problem S_O. Second, when we fix u, we will get a convex optimization problem, which we will call S_{μ_i}. We will see that these sub-problems can be solved efficiently suggesting that an Alternating Minimization or a Batch coordinate descent algorithm Gorski et al. (2007) can be used to solve (6.4). We now provide details about the sub-problems.

Algorithm 8 Alternating Minimization Algorithm

Pick arbitrary starting point μ, initialize u such that $||u||_2 = 1$.
for $t = 1, 2, \cdots, T$ **do**
  Update $\mu \leftarrow \arg\min S_\mu$
  Update $u \leftarrow \arg\min S_O$
end for

6.3.1.3 Subproblem S_O

For a fixed μ, we define $M := \left(\sum_{i=1}^{k} \mu_i x_i x_i^T + \epsilon I\right)^{-1}$. Our model can be written as,

$$\min_{u:||u||_2^2=1} \lambda||\gamma - u||_2^2 + u^T M u$$

Expanding the ℓ_2 penalty term, we get, $||\gamma - u||_2^2 = \gamma^T\gamma - 2u^T\gamma + u^T u = 1 - 2\gamma^T u + u^T u$. The last equality is true since γ is a unit norm eigenvector. So our subproblem is,

$$\min_{u:||u||_2^2=1} u^T(\lambda I + M)u - 2\gamma^T u$$

Note that this is *almost* an eigenvalue problem, that is, we are interested in the largest eigenvalue of $(\lambda I + M)^{-1}$ *except* that we also have the $-2\gamma^T u$ term in the objective. In any case, since the objective is differentiable, we can run a projected gradient method with the projection operator $\Pi_C(\cdot)$ simply corresponds to unit normalization of the vector u at each step, as in Section 2.1.4. Now, observe that,

Observation 4. *When $k = 1$, projection operator is computationally inexpensive and matches the complexity of projection free methods like CG in Algorithm 4.*

Recall that we have already used this observation to tune step sizes or learning rates for CG algorithm with Frobenius norm constraint from empirical performance of SGD, in Section 5.4.1. We will discuss the case when $k > 1$. When the eigenvalue spectrum of the Hessian matrix is large, we should make sure that the top k eigenvectors of the reduced and the full Hessian are close. In this case, we solve,

$$\min_{U \in \mathbb{R}^{p \times k}} \sum_{j=1}^{k} ||\gamma_j - u_j||_2^2 + \text{Tr}\left(U^T M U\right) \quad \text{s.t. } U^T U = I$$

where $u_j, 1 \leqslant j \leqslant k$ is the $j-$th column of U. The main difficulty in running projected gradient algorithm for this problem is that the projection operation is expensive. While in medium scale datasets, projected gradient algorithms tend to be efficient, an algorithm that stays in the feasible set Wen and Yin (2013); Collins et al. (2014) is better suited for large convex problems with orthogonality constraints.

6.3.1.4 Subproblem S_μ

Lemma 6.1. *Denoting the objective function of* (6.4) *as* f, f *is convex w.r.t.* μ *for all* $\mu \in [0,1]^n$.

Proof. The log det term is convex with respect to μ (see ref Boyd and Vandenberghe (2004b)). For a fixed u, denote $M = \sum_{i=1}^{n} \mu_i x_i x_i^T + \epsilon I$. Now the second term can be written as $u^T M^{-1} u$ where $M \succeq 0$, which is a generalized linear over quadratic function and hence is convex with respect to M. Since restriction of a convex function is convex, we have the desired result. □

We briefly note closely related variants of D-optimal design that can be formulated as a SDP in the following remark.

Remark 6.2. *Note that if we use* A*— or* E*—optimal designs instead of the* D*—optimal design, we can reformulate this subproblem* S_μ *as a Semidefinite programming problem with second order cone constraints which can be solved efficiently using standard optimization solvers.*

Corollary 6.3. *Alg.* (8) *constructs a monotonically decreasing sequence of iterates in the objective.*

Synopsis. Even though the geometric formulation mentioned in the previous section provides a clear intuition to the problem formulation, the number of decision variables in (6.3) is $pk + n$ whereas the ED problem for ridge regression (6.1) only had n decision variables. This might become problematic especially when $p \gg n$ which is typically where variable selection is essential. To remedy this issue, we propose an alternative formulation next.

6.3.2 ED-I: Incoherent Experimental Design

Our second formulation utilizes a result related to the LASSO and the well known Restricted Isometry Property (RIP) which we formally define here and then review the statement of a theorem that will be useful in our model.

Definition 6.4 (Restricted Isometry Property Candès and Plan (2009))**.** *Let* $X \in \mathbb{R}^{n \times p}$. *For* $s \geqslant 0$, *the s-restricted isometry constant* δ_s *of* X *is the smallest nonnegative number* δ *such that* $(1-\delta)\|\beta\|_2 \leqslant \|X\beta\|_2 \leqslant (1+\delta)\|\beta\|_2$ *for all* s*—sparse* β, *i.e.,* $\|\beta\|_0 \leqslant s$. *If* $\delta_s < 1$, *then* X *is a* s*—restricted isometry (*s*—RIP).*

With this definition in hand, the next theorem provides a guarantee on the quality of variable selection.

Theorem 6.5. *Candès and Plan (2009) Suppose* X *has* $4s-$*RIP constant* $\delta_{4s} \leqslant \frac{1}{4}$. *Let* $\beta_0 \in \arg\min_\beta\{\|\beta\|_0 : X\beta = y\}$ *and* $\beta_1 \in \arg\min_\beta\{\|\beta\|_1 : X\beta = y\}$. *If* $\|\beta_0\|_0 \leqslant s$, *then* $\beta_1 = \beta_0$.

The theorem suggests that if the matrix X *satisfies* the RIP, then using ℓ_1 instead of ℓ_0 is not a relaxation — the variable selection done by LASSO is exactly equal to that from the ℓ_0 problem. Using this property, we can write a combinatorial form of the ED problem for the LASSO model. More formally, we seek to solve,

$$\arg \max_{\mu\in\{0,1\}^n} \log\det\left(\sum_{i=1}^{n} \mu_i x_i x_i^T + \epsilon I\right)$$
$$\text{s.t. } \mathbf{1}^T\mu \leqslant B, X_{[\mu],:} \text{ is } 4s-\text{RIP}$$

where $X_{[\mu],:}$ denotes the selected subset of rows of X, that is, row i is chosen if $\mu_i = 1$. As before, the objective drives the inclusion of subjects based on the D-optimality criterion. But the constraints require that the data matrix for the selected set satisfy RIP — this will ensure that the variable selection aspect of LASSO works exactly as intended.

Unfortunately, recent results show that checking $4s-$RIP is NP-Hard Bandeira et al. (2012). Whence, even if a black box returns an optimal X, we cannot verify the optimality. As a result, other measures that are easy to check have been developed as surrogates to RIP. We will utilize a common alternative which will lead to a tractable formulation.

Leverage scores. Juditsky and Nemirovski (2011); Drineas et al. (2012) and others have noted that RIP is a strong assumption and in practice, a less conservative requirement may be as effective. As a surrogate, one typically uses *Incoherence* which is easier to compute, this is defined next.

In statistics, the hat matrix and leverage scores determine how much information a data sample carries with respect to the linear model. The hat matrix is defined as $\hat{H} := X(X^TX + \epsilon I)^{-1}X^T$. The leverage score l_i of a particular sample $i \in \{1, .., n\}$ is defined as the $i-$th diagonal element of $\hat{H}$. With each set of leverage scores, we may associate a quantity known as *coherence* defined as $c := \max_i l_i$ where a higher value of c implies that the samples are highly correlated. There are various approaches in

machine learning that use coherence, rather *incoherence*, see Chen et al. (2014) . We can now provide a formulation (analogous to the previous section) that selects the (feasible) set of samples that have the least value of c.

$$\min_{\mu} \log\det\left(\sum_{i=1}^{n}\mu_i x_i x_i^T + \epsilon I\right)^{-1} + \lambda \max_{i=1,\dots,n}\left\{\mu_i e_i^T X\left(\sum_{i=1}^{n}\mu_i x_i x_i^T + \epsilon I\right)^{-1} X^T e_i\right\}$$

$$\text{s.t.} \quad \mu_i \in \{0,1\}, \mathbf{1}^T\mu \leqslant B$$

Observe that since $\mu_i \in \{0,1\} \iff \mu_i^2 \in \{0,1\}, \mu \geqslant 0$, we have an equivalent form of the selection problem as,

$$\min_{\mu} \log\det\left(\sum_{i=1}^{n}\mu_i x_i x_i^T + \epsilon I\right)^{-1} + \lambda \max_{i=1,\dots,n}\left\{\mu_i^2 e_i^T X\left(\sum_{i=1}^{n}\mu_i x_i x_i^T + \epsilon I\right)^{-1} X^T e_i\right\}$$

$$\text{s.t. } \mu_i \in \{0,1\}, \quad \mathbf{1}^T\mu \leqslant B$$

6.3.2.1 Algorithm

We may solve the optimization problem using a randomized coordinate descent method shown in Alg. 9, see Bertsekas and Tsitsiklis (1989) for details. Here, $\Pi_{\mathcal{C}}$ denotes the projection onto $\mathcal{C} := \{\mu : 0 \leqslant \mu \leqslant 1, \mathbf{1}^T\mu \leqslant B\}$.

Algorithm 9 Randomized coordinate descent algorithm for solving (6.5)

```
Pick an arbitrary starting point μ
for t = 1, 2, ··· , T do
  for k = 1, 2, ··· n do
    i ∈ {0, ··· , n},   μ_i ← μ_i − η∇_{μ_i} f
  end for
  Update μ ← Π_C(μ)
end for
```

Lemma 6.6. (6.5) *is a convex optimization problem.*

Proof. Recall that the optimization problem that we seek to solve is given by,

$$\min_{\mu} \log\det\left(\sum_{i=1}^{n}\mu_i x_i x_i^T + \epsilon I\right)^{-1} + \lambda \max_{i=1,\ldots,n}\left\{\mu_i^2 e_i^T X\left(\sum_{i=1}^{n}\mu_i x_i x_i^T + \epsilon I\right)^{-1} X^T e_i\right\}$$
$$\text{s.t. } \mu_i \in \{0,1\}, \quad \mathbf{1}^T\mu \leqslant B \tag{6.5}$$

Introducing auxiliary variable $\nu_i = \mu_i z_i$, where $z_i^t = e_i^T X$ (given) and denoting $M = \sum_{i=1}^{n}\mu_i x_i x_i^T + \epsilon I$ like before, we can rewrite the optimization problem as,

$$\min_{\mu,\nu} \log\det M^{-1} + \lambda \max_{i=1,\ldots,n}\left\{\nu_i^T M^{-1}\nu_i\right\}$$
$$\text{s.t. } \mu_i \in \{0,1\}, \quad \mathbf{1}^T\mu \leqslant B, \quad \nu_i = \mu_i z_i \;\forall\, i = 1,\ldots,n$$

Again using the fact that $\nu_i M^{-1}\nu_i$ is a generalized linear quadratic function for $M \succeq 0$ and that supremum of convex functions is convex, we have that the optimization problem has a convex objective function with a convex feasible set and hence we are done. □

Corollary 6.7. *Denote the objective function of* (6.5) *as* f. *Given an accuracy* $\rho > 0$, *Alg.* (9) *outputs a* $\bar{\mu} \in \mathcal{C}$ *such that* $|f(\mu^*) - f(\bar{\mu})| \leqslant \rho$ *where* μ^* *is the optimal solution.*

The above statements assert that we can find the global optimum of the integrality relaxed problem.

6.3.2.2 Pipage Rounding

Pipage rounding scheme was introduced in Ageev and Sviridenko (2004) to round fractional solutions producing a feasible solution without incurring a substantial loss in the value of the objective function Harvey and Olver (2014), Chekuri et al. (2009). For this technique to work in the present context, we need three conditions to be satisfied:

1. For any $\mu \in \mathcal{C}$, we need a vector ν and $\delta, \tau > 0$ such that $\mu + \delta\nu$ or $\mu - \tau\nu$ have strictly more integral coordinates;

2. For all μ, the objective function f must be convex in the direction of ν; and

3. Most importantly, we need a starting fractional μ with a guarantee that $f(\mu) \leqslant \varpi \cdot \mathbf{opt}$ where **opt** is the optimal value of the (discrete) optimization problem for a known constant ϖ.

With some work (using similar techniques as in Horel et al. (2014)), by choosing a suitable algorithm to solve the relaxed form of (6.5), it is possible to show an approximation ratio for the problem. However, this needs interior point methods and we found that the overall procedure becomes impractical for our datasets. The rounding scheme, if applicable, is still powerful, independent of how the relaxed form of (6.5) is solved.

Lemma 6.8 (Applicability of Rounding)**.** *The objective function in ED-I problem in* (6.5) *satiefies all the three necessary conditions for pipage rounding to preserve the objective value.*

Proof. It is clear from Prop. 6.6 and Cor. 6.7 that conditions (ii) and (iii) are satisfied by the objective f in (6.5). So, we only need to verify condition (i). Suppose that μ is a non-integral vector in $\mathcal{C}$. Also, assume that there are at least two fractional coordinates μ_k, μ_l. Then, let us set $\nu = e_k - e_l$ where e_k, e_l are standard basis vectors in k, l coordinates respectively. Letting $\delta := \min(1 - \mu_k, \mu_l)$ and $\tau := \min(1 - \mu_l, \mu_k)$ we immediately have that $\mu + \delta\nu$ and $\mu - \tau\nu$ are vectors in $\mathcal{C}$ with *strictly more integral coordinates*. Observe that *at least* one of the two vectors stated above is feasible. When both are feasible, if $f(\mu + \delta\nu) \geqslant f(\mu)$, we set $\mu \leftarrow \mu + \delta\nu$, otherwise we set $\mu \leftarrow \mu - \tau\nu$ and repeat until $\mu \in \{0, 1\}^n$. □

Implementation. The rounding procedure described in Lemma 6.8 terminates in at most n steps. Hence the overall complexity scales linearly with number of objective evaluations which involves a determinant and an inverse computation. We now briefly explain the implementation using simple numerical techniques. The key observation is that each step of the procedure involves at most a (symmetric) rank-2 update of $M^* = \sum_{i=1}^{n} \mu_i^* x_i x_i^\top$ where μ_i^* is the output of Alg. (9). If the determinant and inverse of M^* are computed, at each step of the rounding procedure, we need to compute the determinant and inverse of $M^* + \Delta$ where Δ is symmetric and has rank at most 2. Using an inductive technique described in Saigal (1993), the inverse can be updated in $4n^2 + 4n + 1$ operations as opposed to $\mathcal{O}(n^3)$. Then, the update of the determinant only involves computation of four dot products between vectors of length p (Chap. 6 in Nocedal and Wright (2006)).

6.4 Experiments

Data summary. We first evaluated the overall efficacy of our proposed formulations and algorithms on two standard LASSO datasets (*prostate*, Tibshirani (1996) and *lars*, Efron et al. (2004)) and compared their performance to baseline/alternative schemes. After these proof of principle experiments, we performed a set of statistical analyses related to the motivating neuroscience application involving imaging and cognitive data from Alzheimer's Disease Neuroimaging Initiative (ADNI) (*neuro*). These experiments were designed to assess the extent to which statistical power will be compromised when using a smaller fraction of the subjects for estimating linear models associating imaging/clinical covariates with cognitive outcomes. The benchmark data, *prostate* and *lars*, include 8 and 10 features respectively with one dependent variable each, and are well-studied for feature selection. The *neuro* data contains 118 features for image-derived Region-of-Interest (ROI) summaries from Positron Emission Tomography (PET) images. Two cognitive scores were used as dependent variables: Alzheimer's Disease Assessment Score (ADAS) and the diagnostic rating Clinical Dementia Rating (CDR).

Evaluations Setup. The evaluations were two-fold. **First**, we compare the performance of ED-S and ED-I to existing baseline experimental design algorithms including:

1. A random design where a given budget number of instances are selected uniformly at random.

2. A sampling procedure that approximates the distribution of the observations (referred to as "1-mean"), where we first compute the mean of the (current) samples and the standard deviation σ and filter the samples lying inside the ball centered at the mean with radius $\rho||\sigma||_2$ for a given $\rho > 0$. This process is repeated after removing the points lying inside the ball. After k steps, we will be left with k bags of samples with varying sizes. From each of these bags, samples are selected proportionally such that they sum to the budget B.

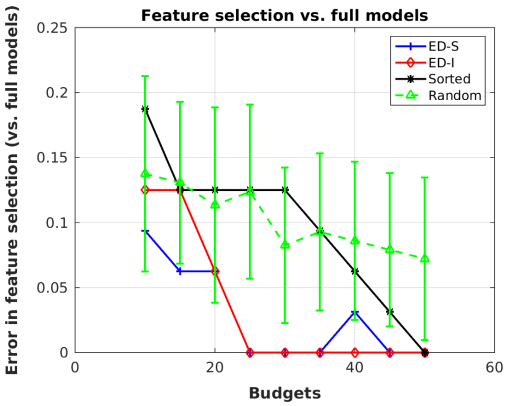
The latter scheme is popular in optimal design where the general idea is to *cover* the dataset by repeatedly selecting 'representer' observations. Our algorithms are compared to these baselines in terms of their ability to consistently pick the correct features (which are defined via the full model i.e., LASSO on *all* instances).

The **second** set of evaluations deal with the model-fit of reduced models (linear models learned using ED-I and ED-S for a given budget) versus the full model. These goodness-of-fit criteria include consistency of zeros and signs of model coefficients, Bayesian Information Criterion (BIC), Akaike information criterion (AIC) and adjusted R^2. Recall that the two linear models we seek to compare do *not* use the same set of observations, and they do not necessarily sparsify the same set of features. Therefore, unlike the classical non-nested hypothesis testing there is no direct way (e.g., using F or χ^2 statistics) to compare them Pesaran and Weeks (2001). To address this problem, we *generate* samples from the full and reduced setups (in a bootstrap sense) and perform a two-sample t-test. The null hypothesis is that the full and reduced setups give 'similar' responses (e.g., ADAS) for a given set of covariates (ROI values). In the ideal case, where the reduced setup captures all the variation of the full one, a *non-significant* (or high) p-value is desired. This is similar to providing "insignificant evidence" against the null (i.e., insignificant evidence that the linear models are different) Berger and Sellke (1987).

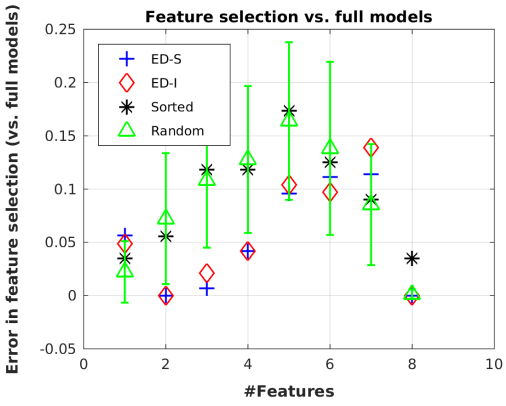
A single workstation with 8 cores and 32GB RAM is used for experiments. We ran 1000 epochs of ED-I, and 50 main iterations of ED-S (with 20 iterations for each of its subproblems). For a fixed budget B, ED-I and ED-S take approximately 7 and 10 min respectively. This is followed by rounding μ_is, and if the rounding generates an infeasible solution (i.e., $\mathbf{1}^T\mu > B$) we randomly drop some of the fractional μ_i subjects. In all the experiments, $\mathbf{1}^T\mu$ overshot B by at most 3 (i.e., most μ_is were binary).

Comparison to baseline designs. Figure 6.2 shows the discrepancy in feature selection versus the baseline algorithms. The y-axis in these plots measures this mismatch where the 'correct' features (i.e., ground truth) are assumed to come from the full LASSO. The x-axis lists different budgets. Clearly, both ED-I and ED-S have consistently smaller errors compared to the two baselines, and achieve zero error in some cases as budget increases (Figure 6.2(a)). We see that the proposed models outperform the 1-mean baseline, although the latter approximates the modes of the data distribution efficiently. Similar behaviour is observed for error plots vs. number of LASSO features (see Figure 6.2(b,d)). We note that the increase in error as the number of LASSO features increases is due to the correlation in the input data. Unlike the baselines, ED-I and ED-S models select the covariates consistently, even when the number of LASSO features is small (left ends of the x-axis).

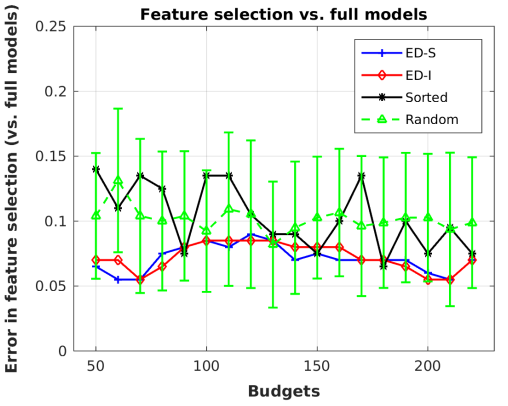
Do reduced models approximate the full model? Figure 6.3 summarizes some



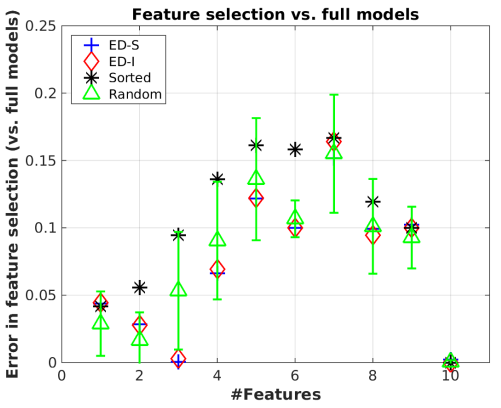
(a) Prostate dataset: error in feature selection vs Budgets



(b) Prostate dataset: error in feature selection vs #Features



(c) Lars dataset: error in feature selection vs Budgets

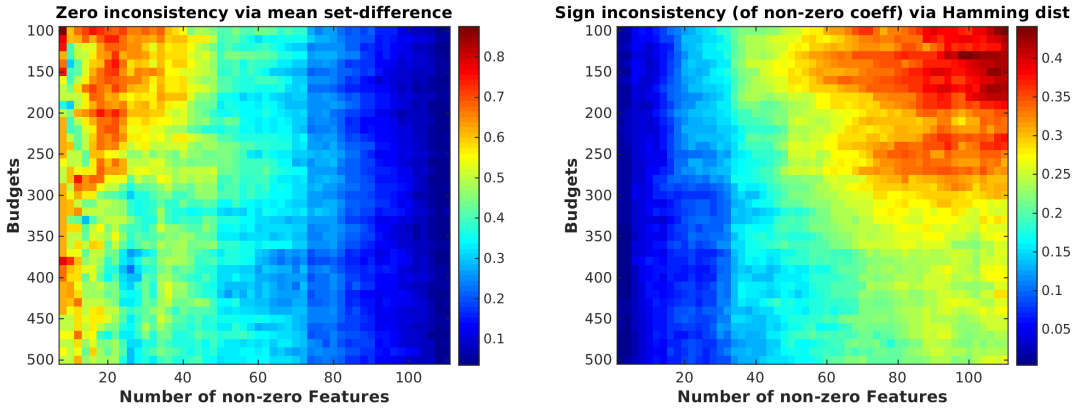


(d) Lars dataset: error in feature selection vs #Features

Figure 6.2: Errors in consistent selection of correct features (derived from full model) for *prostate* (a,b) and *lars* (c,d) datasets.

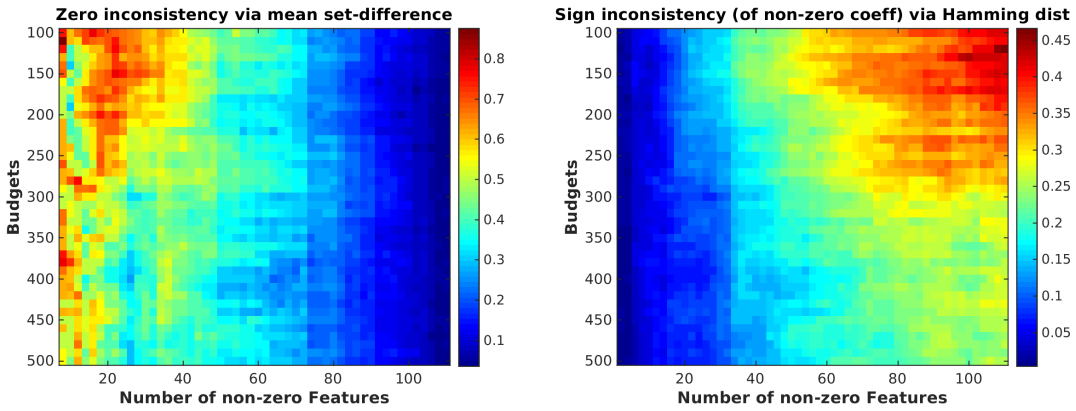
of the model-fit measures comparing ED-I and ED-S to full models on *neuro* data. First observe that the zero inconsistency in Figures 6.3(a,c) decrease gradually as the budget (y-axis) and/or number of allowed nonzero coefficients of LASSO (x-axis) increases. The zero inconsistency refers to the proportion of nonzero coefficients in the reduced setup that are *absent* in the full one. The input ROI features in *neuro* are strongly correlated, and therefore when the number of allowed nonzero coefficients is small (top-left in Figures 6.3(a,c)) LASSO picks few of the many 'similar looking' features making zero inconsistency larger. Such a monotonic trend is also evident for sign inconsistency in Figure 6.3(b,d). However, unlike the previous case, the sign inconsistencies are high for smaller budgets with a large number of nonzero features

(top-right in Figure 6.3(b,d)). This follows directly from the fact that, at the top-right corners LASSO gradually approaches Ridge regression where it is allowed to pick $> 75\%$ of features. Most of these nonzero coefficients will be very small in magnitude, and due to the correlations in the data, the signs of these coefficients are prone to mismatch. The strong linear trends of the inconsistency plots suggest that both ED-I and ED-S are robust to noise and behave well with changing budgets and regularizers.



(a) Zero inconsistency of ED-I

(b) Sign inconsistency of ED-I

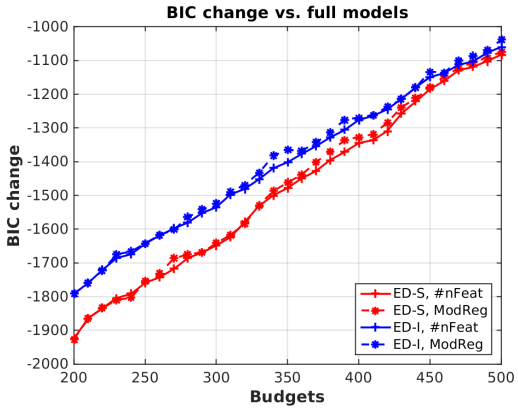


(c) Zero inconsistency of ED-S

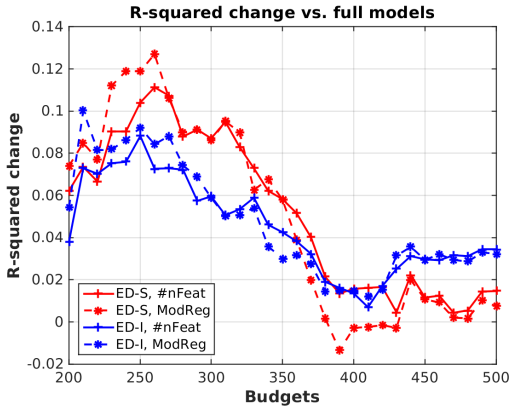
(d) Sign inconsistency of ED-S

Figure 6.3: Zero (a,c) and Sign inconsistency (b,d).

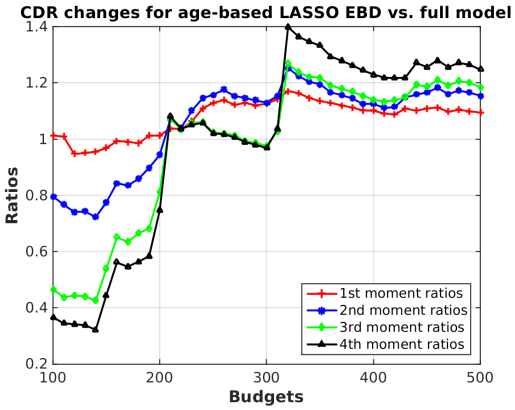
We see in Figure 6.4(a,b) that the reduced setup has much smaller BIC compared to the full one. The red and blue curves correspond to ED-S and ED-I respectively and the plots are averaged across multiple choices of model and LASSO regularizers. Clearly, the magnitude of change decreases monotonically as the budget increases. Further, the adjusted R^2 of reduced setup is larger (Figures 6.4(b)) compared to full ones, although the trends are not as monotonic as was seen for BIC change. This



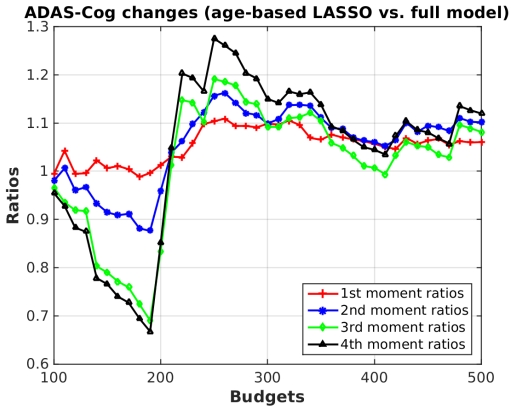
(a) Change in BIC score



(b) Change in R-squared score



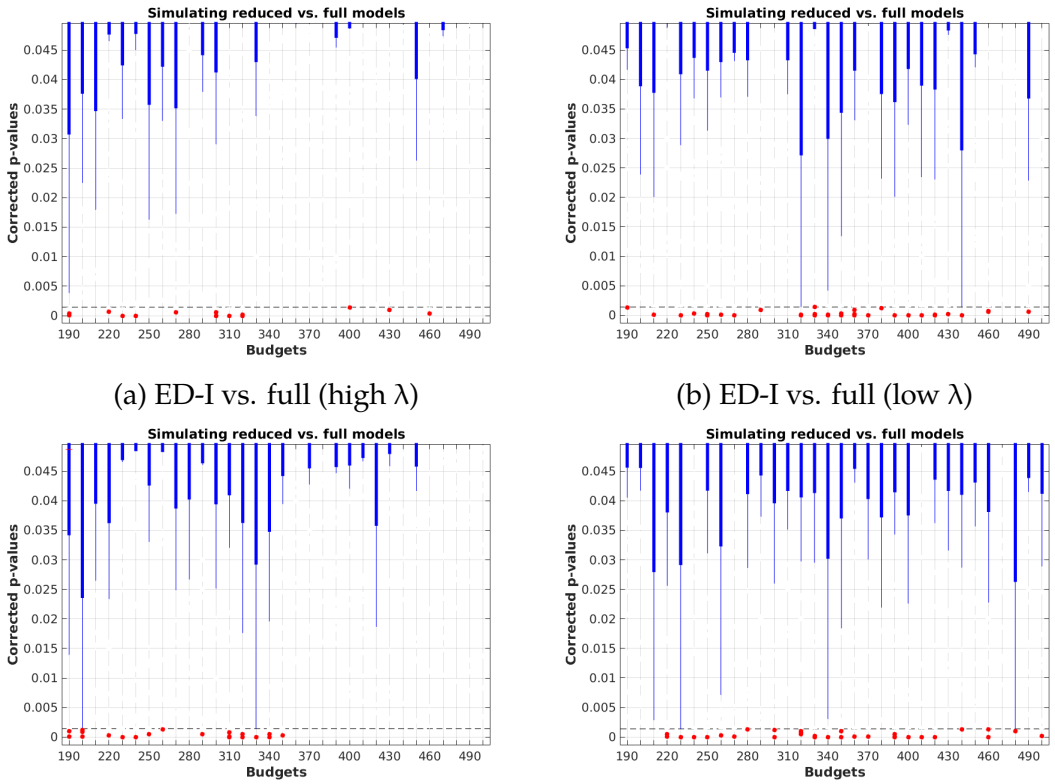
(c) Moment of dependent variable – CDR



(d) Moment of dependent variable – ADAS-Cog change

Figure 6.4: Change in BIC (a) and R^2 (b) vs. full model. (c,d) Dependent variable moments from reduced vs. full models.

implies a better log-likelihood model-fit for reduced setups, which follows from the fact that D-optimality objective of ED-I (and hence ED-S) maximizes the variation among the selected subjects. Any input feature and/or dependent variable noise (like corrupted observations, sampling noise) from the unselected subjects (which are now linear combinations of the selected ones), does not propagate into the linear model estimation. This interpretation is also supported by noticing that the gain in R^2 is higher for smaller budgets and reduces as the budget increases (Figure 6.4(b)). Note that, the trend in Figure 6.4(b) also implies that the optimal choices of budgets for the *neuro* dataset are ~ 400 ($\sim 40\%$ of the total population). The R^2 change of reduced vs. full model (6.4(b)) was used to pick a "good" budget. Indeed, full dataset is always better than the reduced, but here we refer to the smallest budget that approximates



(a) ED-I vs. full (high λ)

(b) ED-I vs. full (low λ)

(c) ED-S vs. full (high λ)

(d) ED-S vs. full (low λ)

Figure 6.5: Quantifying the difference between reduced and full models using p-values obtained after hypothesis testing.

the full model in R^2 change as "optimal". Figures 6.3, 6.4 also show that the two proposed algorithms (ED-I and ED-S) yield very similar results.

Building upon these model-fit measures Figure 6.4(c,d) shows the ratios of 1^{st} to 4^{th} moments of the samples *generated* from the full and reduced setups. Most of these ratios (even for higher order moments) are centered around 1, suggesting that reduced setups are *excellent* approximations of the full one. We also verified that the same conclusion holds for other properties of interest for sparse models such as mismatch of selected subjects between ED-I and ED-S and sparsistency of the algorithms. We also quantify the observations in Figure 6.4(c,d) by performing hypothesis testing of the reduced vs. full models. This is shown in the box plots of p-values in Figure 6.5(a,b) and 6.5(c,d) for ED-I and ED-S respectively for different budgets (x-axis). Recall that the null hypothesis is that the samples from the two setups are *similar*. Since we perform multiple testing (tests across different model and LASSO regularizers) for each budget, the p-values are Bonferroni corrected (the dotted line in each plot). The

'red' dots *below* this Bonferroni threshold are the significant p-values implying that the samples from reduced and full models are different. Clearly, these significant ones are *very few* and scattered across all budgets, and much smaller compared to the non-significant ones (denoted by blue boxes). This means that the number of budget and regularizer combinations that reject the null is extremely small. Also, such cases are much smaller for ED-I (Figure 6.5(a,b)) compared to ED-S (Figure 6.5(c,d)). Note that depending on the number of samples (for computing t-statistics), these scattered red points will further reduce. Overall, we see that the reduced setups capture all modeling/distributional characteristics of the full one for almost all choices of budget, LASSO and/or λ's.

6.5 Conclusions

We addressed the problem of experimental design in sparse linear models common in many applications. We proposed two novel formulations and derived efficient algorithms for experimental design problems on a budget. We presented detailed analysis along with the optimization schemes for ℓ_1 regularized linear models. Our technical results hold for a more general class of sparse linear models as well as optimal design criteria other than D−optimality (as long as the relaxation yields a convex model). We showed an extensive set of experiments providing strong evidence for the robustness and efficiency of these formulations. The ideas described here have applications in experiment design problems in neuroscience leading to potential cost savings in longitudinal studies aimed at clinical trials. The code of both ED-I and ED-S is publicly available here.

Chapter 7

Robust Blind Deconvolution via Mirror Descent

In this chapter, we develop robust and efficient method for the blind deconvolution problem. The proposed method can be seen as the classical gradient descent method (in Section 2.1.2) where the metric may be specified using the geometry of the constraint set. We analyze the convergence, robustness behavior of our PRIDA Algorithm 10 and its empirical behavior (on standard datasets).

7.1 Introduction

Image deblurring has been an active area of study in computer vision for nearly five decades. The early proposals sought to sharpen or *deblur* images from photographs by relying on parameters relating the exposure and the amplifier gain, e.g., via the use of the Stroke/Zech division filter Stroke and Halioua (1970). Most contemporary algorithms for deblurring, however, pose the problem as blind deconvolution, which refers to separating a true unknown signal and some unknown “kernel" or “filter" when provided knowledge *only* of the noisy measurement of the signal convolved with the filter. This is a fundamental topic today in signal processing and vision, and remains challenging due to its non-convex and ill-posed nature — only within the last few years has brisk progress been made towards methods that gracefully handle real images encountered in practice Levin et al. (2009); Campisi and Egiazarian (2016). These recent developments notwithstanding, due to the foregoing technical challenges, we are often unable to guarantee provably good solutions to the underlying

optimization task, and strategies to address these issues are being studied by various researchers in our community today Perrone et al. (2015); Jin et al. (2017); Li et al. (2016b); Campisi and Egiazarian (2016).

7.1.1 Related Work

Modern approaches generally prefer one of two related but distinct strategies for blind deconvolution. On the statistical side, research has primarily revolved around Bayesian methods Ruiz et al. (2015), taking advantage of useful priors ranging from fundamental image geometry in the context of its relation to edge detection and saliency, to expert knowledge of the specific application domain of interest Cho and Lee (2009); Xu and Jia (2010). While these ideas provide guarantees in terms of robustness, the development of efficient sampling (e.g., Gibbs sampler) and inference algorithms remains an active topic of research. On the optimization side, total variation regularization has proven to be extremely effective in general image deblurring Perrone and Favaro (2014); Chan and Wong (1998); Osher et al. (2005) in a variety of image domains. While the mathematical properties of total variation have been well studied in applied mathematics, signal processing and machine learning, our understanding of the robustness and convergence behavior of even the best performing algorithms for blind deconvolution based on this construct remains limited, although there is exciting progress being made Srinivasan et al. (2017). A primary motivation of our work is to shed light on these theoretical issues.

7.1.2 Deep Learning Approaches

Separate from, but complementary to the above lines of work, the enormous success of deep convolutional architectures in vision has led to a number of papers Sun et al. (2015); Schuler et al. (2016); Chakrabarti (2016); Noroozi et al. (2017) exploring how such successes can be adapted to deconvolution in general. While some initial attempts showed the use of machine learning methods for *non-blind* image deconvolution (i.e., the blur kernel is provided), discriminatively trained architectures have now been shown to work quite well for the general setting, both with and without priors on motion blur types. A natural question one may ask is whether an in-depth study of the core blind deconvolution formulation and its properties is relevant in light of this still evolving body of convolutional neural networks based literature. The

reader will see that our work is complementary. Of the recent proposals in this line of work reformulate deconvolution as a supervised learning problem by synthesizing blurred and sharp image pairs, and are often based on some form of blind deconvolution sub-routine internally Schuler et al. (2016). As these methods get closer to practical deployment in mission critical applications, a detailed assessment of their behavior profile will be a first order requirement for regulation compliance. To enable investigating the robustness and convergence properties of these architectures and their resilience to adversarial examples — as is happening in the last few years for other problems in both computer vision and machine learning Su et al. (2017); Moosavi-Dezfooli et al. (2016); Moosavi Dezfooli et al. (2017) — we will necessarily rely on and benefit from a "first principles" understanding of such properties for the standalone (i.e., shallow regimes of) blind deconvolution.

Our Results. We provide (1) a *quantifiably and provably* **robust** algorithm for blind deconvolution with (2) **guaranteed convergence** properties. To our knowledge, no algorithm is currently known that offers *both* these properties at once. Our convergence guarantees match the best known results in optimization at this time. Our technical analysis is also backed up by practical performance. Via an extensive experimental study, we show that on most available benchmarks, our simple algorithm competes favorably with (or is superior to) the state of the art, and provide a user-friendly implementation which can be easily extended to a complete user-interactive deblurring package.

7.1.3 Estimation Methods

Methods for image deblurring via blind deconvolution have employed a variety of regularizations derived from a wide range of image priors. The literature is vast and so we restrict our discussion to a subset of works that are closely related to or motivate our proposed strategy. The earlier forms of regularization were based on the ℓ_2-norm in You and Kaveh (1996), where an alternating minimization scheme was proposed. More recent improvements have been proposed by Cho and Lee Cho and Lee (2009) and Xu and Jia Xu and Jia (2010). On the other hand, total variation regularization – the defacto choice in many state of the art methods today – was initially deployed in image denoising applications Rudin et al. (1992); Vogel and Oman (1996). and brought to the image deconvolution problem by Chan and Wong Chan and Wong (1998). A nice result by Osher et al. (2005) gives a variational iterative procedure for solving the

total variation objective. A conceptually distinct set of results for blind deconvolution adopt a more statistical approach instead. Levin et al. in Levin et al. (2009) provide analysis of algorithms following *maximum a posteriori* (MAP) estimators. A recent work Ruiz et al. (2015) gives a nice and comprehensive overview of Bayesian methods for blind deconvolution. A few years back, Perrone and Favaro (2014) built on analysis in Levin et al. (2009) and demonstrated experimentally the behavior of Chan and Wong (1998). In a follow-up work, those authors showed the advantage of a logarithmic prior Perrone et al. (2015), obtaining state of the art results with a mild modification to the classical TV-norm based formulation which we will present shortly. Separate from total variation regularization based approaches, interesting results have been shown by Michaeli and Irani (2014) through an ℓ_0 regularization on text images and by Michaeli and Irani (2014); Sun et al. (2013) via the use of patch priors. Recently, a detailed comparative study was conducted by Lai et al. (2016), in which participants were asked to qualitatively compare two results from multiple algorithms, a subset of which are described in our review above. (Jin et al., 2018) proposes a size normalized formulation and develops efficient alternating minimization algorithm to solve the formulation.

In the last few years, ideas based on specialized deep networks have started yielding interesting results for this problem (Jin et al., 2019; Tao et al., 2018). For example, Sun et al. (2015) was among the first approaches for motion blur removal by posing the problem as a supervised learning task and training a convolutional neural network (CNN) to infer the parameters. Schuler built on these results in Schuler et al. (2016), and Chakrabarti Chakrabarti (2016) constructed a network to predict the Fourier coefficients of the filter necessary to deblur specific image patches. Taking advantage of modern convolutional architectures, Nah et al. (2016) constructed deep multi-scale networks for dynamic scene deblurring with strong empirical results. Generative Adversarial Networks have also been applied with measured success Ramakrishnan et al. (2017).

7.2 The Blind Deconvolution problem

Throughout this chapter we assume that an image is an $n-$dimensional vector taking values between 0 and 1 without loss of generality. We will use $f \in \mathbb{R}^n$ and $k \in \mathbb{R}^s$ to denote the vectorized sharp image and blur kernel, both of which are to be estimated

given the vectorized blurry image $b \in \mathbb{R}^n$. Mathematically, the model can be written as,

$$b = f * k + \alpha,$$

where $*$ denotes the (discrete circular) convolution between two signals and α denotes the independent noise vector at each pixel. Assuming that $\alpha_i \sim \mathcal{N}(0,1)\ \forall i \in [n]$, we can estimate f, k by maximizing the log-likelihood, corresponding to solving the following least squares optimization problem,

$$\min_{f,k} \|f * k - b\|_2^2. \tag{7.1}$$

Observe that the number of parameters to be estimated is $n+s$ and can be much larger than the number of observations n if the kernel is large. To solve for solutions to (7.1), many regularization functions (or priors) and/or constraints have been proposed in the literature Levin et al. (2009); Ruiz et al. (2015); Campisi and Egiazarian (2016). To keep the presentation simple, we will focus our attention on two generic components that have shown strong empirical performance to specify the full model.

- **Component 1.** The Total Variation (TV) ℓ_p-norm on f has been shown to promote smoothness of the estimated image Chambolle and Lions (1997). The image TV norm is defined as some norm of its discrete gradient field over the image lattice $(\nabla_i f(\cdot), \nabla_j f(\cdot))$:

  $$\|f\|_{TV}^p := \left(\left(\|\nabla_i f\|_p + \|\nabla_j f\|_p \right)^p \right.$$

  Note that for $p \in \{1, 2\}$, this corresponds to the classical anisotropic and isotropic TV norm respectively. Our theoretical analysis extends to any $p \geqslant 1$, but we will assume that $p = 2$ to describe our results.

- **Component 2.** Recall from (previous) Chapters 3, 5, and 6 that the constraints were informed by the needs of the application or domain knowledge. We will follow the same approach here. That is, in order to define a reasonable constraint set, we appeal to the fundamentals of the image capture process. Pixel values are explicitly a positive function of the photon count at a specific point on the image sensor, and so we enforce the constraint that the kernel must be nonnegative. Further, a blurred image can be interpreted as a weighted average

of a sharp image captured with slight shifts, typically stemming from an extended exposure time due to a variety of reasons. Together, these requirements form our constraint set: the probability simplex $\Delta := \{x \geqslant 0 : 1^\top x = 1\}$.

With these two pieces, the problem that we aim to solve can be formally written as,

$$\min_{f,k \in \Delta} \mathcal{L}(f,k) := \|f * k - b\|_2^2 + \lambda \|f\|_{TV} \tag{7.2}$$

where $\lambda \geqslant 0$ is a tunable regularization parameter. Intuitively, higher values of λ will encourage more smoothness in the optimal sharp image f of (7.2).

7.3 The PRIDA Algorithm

In principle, Problem (7.2) should be easily amenable to many continuous optimization methods but in practice, Perrone and Favaro (2014) provides compelling evidence that choosing the right algorithm is critical to a successful recovery of the sharp image f. Notice two important but straightforward properties of the optimization in (7.2):

1. the objective is smooth and convex in each argument f, k individually but *not* jointly convex and
2. the feasible set Δ is convex and compact.

Roadmap. We will see shortly that properly exploiting these two simple properties will suggest a natural choice of an algorithm that is familiar in non-linear optimization but not very broadly used in machine learning and vision. Interestingly, after we motivate the choice of the algorithm, we will see how the properties above provide certain technical results that yield guarantees for fast convergence rates and subsequently, suggest strategies for a rigorous robustness analysis. But first, let us analyze why some obvious simplifications and/or direct use of an alternating scheme may not be an effective strategy for this model.

Potential Idea: ignore nonconvexity? A natural strategy to solve (7.2) may be to use an algorithm which exploits the convexity of $\mathcal{L}$ individually with respect to f and k. A well known method that offers this capability is the Alternating Minimization (AM) algorithm Hardt (2014). The AM algorithm for this model performs the following

calculation (or update) at each iteration:

$$f^{t+1} \leftarrow \arg\min_{f} \mathcal{L}\left(f, k^{t}\right) \tag{7.3}$$

$$k^{t+1} \leftarrow \arg\min_{k \in \Delta} \mathcal{L}\left(f^{t+1}, k\right). \tag{7.4}$$

Since both the subproblems (7.3) and (7.4) are convex optimization problems, they can both be computed efficiently Hardt (2014).

A potential problem of Alternating Minimization: Random versus structured blur. There are some recent results that analyze the convergence behavior of the AM algorithm for *random* blur kernels Hardt (2014), and offer guarantees on its performance. Unfortunately, it is still an open question whether such guarantees are available for *structured blur kernels* that we universally encounter in vision. In fact, Perrone and Favaro (2014) explicitly constructs an illustrative example where the AM algorithm converges to a strict saddle point due to the nonconvexity of $\mathcal{L}$.

In the context of the blind deconvolution problem, strict saddle points correspond to a **no blur solution**, that is, when the kernel k has *only one nonzero entry*. We see that in Perrone and Favaro (2014) (cf. Section 3.4), the authors give a clear example where the AM algorithm converges to the no blur solution, and thereby propose specific work-arounds to solve the subproblem (7.4) such that the algorithm empirically converges to the desired one instead. The authors also show that their scheme performs consistently better on many standard benchmark datasets. However, to our knowledge, it is not clear if the procedure suggested in Perrone and Favaro (2014) guarantees convergence in general. Whether the method in Perrone and Favaro (2014) provably returns a minimizer of (7.2) is also not described in their work.

Revisit Gradient Methods? Instead of the alternating scheme, we take a more "classical" approach to this problem and propose updating both f and k simultaneously at each iteration. Our choice of algorithm, described shortly, is motivated by **two key insights** in Problem (7.2).

1. **First**, for a smooth optimization problem, it has recently become known that the set of initial points from where a first order gradient method converges to a saddle point has a Lebesgue measure of zero Panageas and Piliouras (2016). This immediately entails that with with very high probability, a gradient method will converge to a local minimizer.

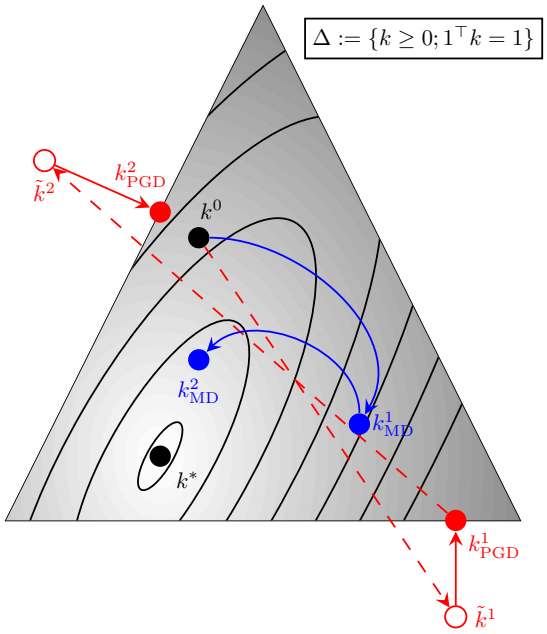


Figure 7.1: Visualization of Projected Gradient Descent (PGD) and Mirror Descent (MD) on the probability simplex. If the step size is not selected carefully, projected updates are more likely to find solutions along the boundary.

2. **Second**, the geometry of Δ will allow us to provably speed up the convergence which is interesting from both a theoretical standpoint and a practical one.

A Mirror-descent style algorithm. To describe our algorithm, it is easiest to briefly review the form of a classical mirror descent (MD) scheme used in convex optimization. Recall that the standard way to solve constrained optimization problems is to use projections, that is, we first take a (negative) gradient step and then a *Euclidean* projection on to the feasible set, assuming that this is easy to do (as is the case with norm balls, hyperplanes and so on). Recall from Section 2.1.4 that this procedure is the Projected Gradient Descent (PGD):

$$x^{t+1} = \Pi_\Delta(x^t - \eta_x g_x^t)$$

where Π_Δ is the Euclidean projection. Under mild conditions on the step size η_x, PGD in fact guarantees convergence. However, the use of PGD type algorithms require some caution since PGD completely disregards the geometry of the feasible set and only uses the local behavior of the objective function. That is, the objective

Algorithm 10 Provably Robust Image Deblurring Algorithm (PRIDA)

Pick a starting point $k^0 \in \Delta$, $f^0 \in [0,1]^n$.
for $t = 0, 1, 2, \cdots, T$ **do**
$(g_f^t, g_k^t) \leftarrow (\nabla \mathcal{L}_f(f^t, k^t), \nabla \mathcal{L}_k(f^t, k^t))$
$f^{t+1} \leftarrow f^t - \eta_f g_f^t$
$\hat{k}_i^{t+1} \leftarrow k_i^t \exp\left(-\eta_{k_i}^t g_{k_i}^t\right) \quad (*)$
$k^{t+1} \leftarrow \frac{\hat{k}^{t+1}}{\sum_{i=1}^{s} \hat{k}^{t+1}}$
end for

function in the projection operator $\Pi_C(\cdot)$ (in Definition 2.16) treats all points in C equally. This is not always desirable since solutions near the center of C are always more preferable, irrespective of the objective function f. Hence, the algorithm can be very inefficient particularly in the high dimensional and large scale settings we see in vision Mahadevan and Liu (2012); Luong et al. (2012).

Intuitively, Mirror Descent (MD) addresses this problem with the following simple modification: it is better to choose a function that acts like a metric *depending on the feasible set*. This function is called the Distance Generating Function (DGF) and moreover, it is enough for that function to be a metric just on the feasible set Juditsky et al.. Exploiting this property, MD has been used to design algorithms that are provably faster than PGD Nesterov (2005) and is the preferred algorithm in many applications Srebro et al. (2011); Jain and Thakurta (2012). An excellent description of the MD algorithm and its variants is given in Nemirovski (2012). Recently, Zhou et al. (2017) showed how to extend MD to a class of nonconvex problems called variationally coherent problems. But unfortunately, the results in (7.2) are not applicable to our problem.

Motivated by the above discussion, we propose a Provably Robust Image Deconvolution Algorithm (PRIDA), shown in Alg. 10. As alluded to previously, PRIDA is similar in spirit to the MD algorithm in Convex Optimization. The main difference between the standard MD algorithm and PRIDA is that the step size is chosen independently for each coordinate. The intuition behind the step size rule can be seen as follows: if a coordinate j of the filter (kernel) k at the $t-$th iteration k_i^t is large in magnitude, then we expect it to remain reasonably high at the $t+1-$th iteration. Our empirical results show that this is very effective in practice. Next, we show that PRIDA converges provably to a minimizer.

7.4 Convergence, Robustness, and More

To analyze PRIDA Algorithm 10, we first write the following equivalent interpretation of the update steps:

$$f^{t+1} = f^t - \eta_f g_f^t = \arg\min_f \langle g_f^t, f\rangle + \frac{1}{2\eta_f}\|f - f^t\|_2^2$$
$$\hat{k}^{t+1} = k_i^t \exp\left(-\eta_{k_i}^t g_{k_i}^t\right) = \arg\min_{\hat{k}} \langle \eta_k^t \circ g_k^t, \hat{k}\rangle + \mathrm{KL}\left(\hat{k}\|k^t\right) \tag{7.5}$$
$$k^{t+1} = \frac{\hat{k}^{t+1}}{\sum_{i=1}^s \hat{k}^{t+1}} = \arg\min_{k\in\Delta} \mathrm{KL}\left(k\|\hat{k}^{t+1}\right) \tag{7.6}$$

where $\mathrm{KL}(x\|y)$ represents the usual Kullback-Leibler divergence between x and y, $\langle\cdot,\cdot\rangle$ is the inner product, and $\circ$ denotes element-wise multiplication. Note that when the KL divergence function is replaced by the Euclidean norm, the algorithm becomes the standard PGD update. Observe that $\mathrm{KL}(x\|y)$ acts as a distance-generating function on simplex Δ, and hence k^{t+1} is unique. In order to show convergence we use the following intermediate result.

Lemma 7.1. *Let* $x^* = \arg\min_{x\in\Delta}\langle z, x\rangle + \mathrm{KL}(x\|x^0)$. *Then for any* $y \in \Delta, z \in \mathbb{R}^s$, *we have that*,

$$\langle z, y\rangle + \mathrm{KL}(y\|x^0) \geqslant \langle z, x^*\rangle + \mathrm{KL}(x^*\|x^0) + \mathrm{KL}(y\|x^*).$$

Proof. Define $G(x) := \langle z, x\rangle + \mathrm{KL}(x\|x^0)$. Since x^* minimizes G over Δ, and $G(x)$ is differentiable and strongly convex on Δ with respect to ℓ_1-norm Shapiro et al. (2009), the gradient g at x^* should satisfy the following inequality,

$$\langle g, x - x^*\rangle \geqslant 0 \;\forall\, x \in \Delta.$$

Now from the property of gradient and convexity of the linear function, we have that,

$$\langle z, y\rangle \geqslant \langle z, x^*\rangle + \langle z, y - x^*\rangle$$

Plugging in the derivative of KL divergence, adding and subtracting $\sum_i x_i^* \log x_i^* + y_i \log y_i + x_i^0 \log x_i^0$, and rearranging gives the desired result. □

With this in hand, we have the following convergence result.

Theorem 7.2. *Let* $\|\nabla^2\mathcal{L}\| \leqslant L$, *then with step sizes* $\eta_f^t, \eta_{k_i}^t \leqslant \min\left(\frac{\alpha\|k\|_\infty}{k_i^t\|g_k\|_\infty}, \frac{1}{L}\right)$ *where* $1 > \alpha > 0$ *is fixed, PRIDA converges to a local minimizer of* $\mathcal{L}$ *(avoids strict saddle points) almost surely.*

Proof. We will assume without loss of generality for the analysis that the step size $\eta = \eta_f^t = \eta_{k_i}^t = 1/L$. We prove the convergence in two steps. In step 1, we show that the iterates of the PRIDA algorithm 10 converges to a fixed point. In step 2, we show that there is a subsequence that converges to a stationary point, that is, a point that approximately satisfies the first order necessary conditions. We then use Lee et al. (2017) to show that such a stationary point is a locally optimal solution.

Step 1. For notational convenience, let $p := [f, k]$, where the first n coordinates denote f and the last s coordinates denote k respectively. Define $h(p) := \mathcal{L}(p) + I_\Delta(p)$ where $I_\Delta(p)$ is the indicator function that takes the value 0 if $p_k := p_{[n+1:n+s]} \in \Delta$ and ∞ otherwise. Then for any $x, y \in \mathbb{R}^{n+s}$, we have that,

$$
\begin{aligned}
h(x) = \mathcal{L}(x) + I_\Delta(x) &\leqslant \mathcal{L}(y) + \nabla\mathcal{L}(y)^T(x-y) + \frac{L}{2}\|x-y\|_2^2 + I_\Delta(x) && (7.7)\\
&= \mathcal{L}(y) + \nabla\mathcal{L}(y)^T(x-y) + \frac{L}{2}\|x_f - y_f\|_2^2 + \frac{L}{2}\|x_k - y_k\|_2^2 + I_\Delta(x)\\
&\leqslant \mathcal{L}(y) + \nabla\mathcal{L}(y)^T(x-y) + \frac{L}{2}\|x_f - y_f\|_2^2 + \frac{L}{2}\|x_k - y_k\|_1^2 + I_\Delta(x) && (7.8)\\
&\leqslant \mathcal{L}(y) + \nabla\mathcal{L}(y)^T(x-y) + \frac{L}{2}\|x_f - y_f\|_2^2 + \frac{L}{4}KL(x_k\|y_k) + I_\Delta(x) =: u(x,y) && (7.9)
\end{aligned}
$$

where (7.7) is by smoothness of the gradient (assumption), (7.8) is by Cauchy-Schwarz inequality and (7.9) is by Pinsker's inequality (see page 88 in Tsybakov (2008)). Note that the minimizer of $u(x, y)$ with respect to x exactly corresponds to the update rule in PRIDA and that $u(x, y)$ is strongly convex in x (again due to Pinsker's inequality, see page 301 in Bubeck et al. (2015)). Hence we can bound the per iteration improvement by,

$$
\begin{aligned}
h(p^t) - h(p^{t+1}) \geqslant h(p^t) - u(p^{t+1}, x^t) &= u(p^t, p^t) - u(p^{t+1}, p^t)\\
&\geqslant \frac{L}{2}\|p_f^{t+1} - p_f^t\|_2^2 + \frac{L}{4}KL(p_k^{t+1}\|p_k^t) \geqslant \frac{L}{2}\|p^{t+1} - p^t\|_2^2 \qquad (7.10)
\end{aligned}
$$

where inequality (7.10) follows from (7.9) . Summing up the inequalities (over t) in (7.10), we see that the $\lim_{t\to\infty}\|p^{t+1} - p^t\| = 0$, that is, algorithm converges to a fixed point.

Step 2. Since the update rule for f is standard gradient descent, we know that the iterates converge to a point where the gradient vanishes, see section 1.2.3. in Nesterov (2013). So we focus on the update rule for the k for which we use Lemma 7.1. Taking $z = \eta\left(\mathcal{L}\left(p^t\right) + g_k^{t^T}\left(p_k - p_k^t\right)\right)$ there, we have that,

$$\begin{aligned} KL(p_k^*\|p_k^{t+1}) &\leqslant KL(p_k^*\|p_k^t) + \eta g_k^{t^T}(p_k^* - p_k^{t+1}) - KL(p_k^{t+1}\|p_k^t) \\ &= KL(p_k^*\|p_k^t) + \eta g_k^{t^T}(p_k^* - p_k^t) + \eta g_k^{t^T}(p_k^t - p^{t+1}) - KL(p_k^{t+1}\|p_k^t) \\ &\leqslant KL(p_k^*\|p_k^t) + \eta g_k^{t^T}(p_k^* - p_k^t) + \mathcal{L}(p^t) - \mathcal{L}(p^{t+1}) \end{aligned} \tag{7.11}$$

where we used the smoothness assumption in (7.11). Again summing the inequalities in (7.11) (over t), we have that,

$$\begin{aligned} KL(p_k^*\|p_k^{t+1}) \leqslant & KL(p_k^*\|p_k^0) + \eta \sum_t g_k^{t^T}(p_k^* - p_k^t) + \mathcal{L}(p^0) \\ \implies T \min_t g_k^{t^T}(p_k^t - p_k^*) \leqslant & \sum_t g_k^{t^T}(p_k^t - p_k^*) \leqslant (KL(p_k^*\|p_k^0) + \mathcal{L}(p^0))L. \end{aligned}$$

Taking the limit as $T \to \infty$, we showed that we can find a point that satisfies the first order optimality conditions of our optimization problem. Thus we have shown that after T steps, we can find a point that is $O(1/T)$ optimal. PRIDA iterates now satisfy the assumptions of Proposition 10 in Lee et al. (2017), and so by Corollary 7 therein it directly follows that PRIDA does not converge to a strict saddle point almost surely. □

While we can get the same convergence rate (up to logarithmic factors in s) of $O(1/T)$ as that of PGD (see Ghadimi and Lan (2016)), the efficiency of PRIDA comes from the fact that each iteration of PRIDA takes $O(s)$ time, compared to the $O(s \log s)$ required in PGD (see Chen and Ye (2011) for details) and is trivially parallelizable/amenable to GPU implementation.

7.4.1 Robustness

Having shown the convergence of PRIDA, the natural follow-up investigation is to characterize its behavior in terms of its noise tolerance. We call an algorithm robust if it produces the same output on two different images such that one of them is a slightly perturbed version of the other. This notion of robustness has been recently introduced

in the machine learning literature under the context of algorithmic stability Hardt et al. (2015) and adversarial attacks (Samangouei et al., 2018). Recent results in our community show that this is a critically desirable property of algorithms used in vision-based deployments since they are often sensitive to very small perturbations Moosavi Dezfooli et al. (2017); Su et al. (2017).

Plan of Attack. Using only the main concepts of stability, we aim to measure the robustness of our algorithm. In typical stability analyses, noise is often introduced in the gradient computation, as a proxy for stochastic or approximate gradient updates. We follow this idea, and aim to bound the difference between the result of a noisy gradient update and a clean one. To be specific, we assume that two images, one with noise and the other without, produce gradients that are approximately the same.

Hence, at iteration t we observe some noisy gradient $\tilde{g}_k^t$ of g_k^t (and respectively $\tilde{g}_f^t$ of g_f^t). We would like to bound the distance between k^{t+1} and $\tilde{k}^{t+1}$ (f^{t+1} of $\tilde{f}^{t+1}$). In what follows, we look only at the update for the kernel k, but note that an analogous argument can be made for the sharp image f: the update step for f is essentially a (sub)gradient step, and so the argument is simpler.

Lemma 7.3. *Let k^0 be the initial point where all coordinates are equal. Let $g := g_k^1$ be the true gradient and $h := g_k^1 + \alpha$ be some noisy gradient. Then, we have that k_g^1, k_h^1 computed using g and h are δ-close in the ℓ_1 sense.*

Proof. In order to study the robustness properties of our algorithm, we will use the interpretation of PRIDA given in (7.5) and (7.6). Because the noisy gradient is only being used in the (7.5), we analyze how much iterates can stray after each of the two updates separately. To that end define the intermediate iterate $x := \hat{k}_g^1$ computed using the true gradient and similarly $y := \hat{k}_h^1$ the noisy one. To make the proof simple, we will assume that the step size is same for all the coordinates, that is, $\eta_i^1 \equiv \eta$ (say $1/L$) and note that the argument can be easily extended for the general case. Then, the distance between x and y can be bounded as follows,

$$\begin{aligned}\|x - y\|_1 = \sum_i^n \left| k_i^0 e^{-\eta g_i} - k_i^0 e^{-\eta(g+\alpha)_i} \right| &= \sum_i^n k_i^0 e^{-\eta g_i} \left|1 - e^{-\eta \alpha_i}\right| \\ &\leqslant \sum_i^n k_i^0 e^{-\eta g_i} |\eta \alpha_i| \leqslant \eta \bar{\alpha} \|x\|_1 \end{aligned} \tag{7.12}$$

where the first step follows from the definition of x and y, and the last two from the fact that $e^{-x} \geqslant 1 - x \; \forall x$ and $\bar{\alpha} := \max_i |\alpha_i|$. Now we show that the second step of the update, which corresponds to a simple normalization, is also well behaved:

$$\begin{aligned}
\left\| \frac{x}{\|x\|_1} - \frac{y}{\|y\|_1} \right\|_1 &= \left\| \frac{x}{\|x\|_1} - \frac{y}{\|x\|_1} + \frac{y}{\|x\|_1} - \frac{y}{\|y\|_1} \right\|_1 \\
&\leqslant \left\| \frac{x}{\|x\|_1} - \frac{y}{\|x\|_1} \right\|_1 + \left\| \frac{y}{\|x\|_1} - \frac{y}{\|y\|_1} \right\|_1 \qquad (7.13) \\
&= \frac{\|x - y\|_1}{\|x\|_1} + \left\| \frac{y \left(\|y\|_1 - \|x\|_1 \right)}{\|x\|_1 \|y\|_1} \right\|_1 \\
&= \frac{\|x - y\|_1}{\|x\|_1} + \frac{\left| \|y\|_1 - \|x\|_1 \right|}{\|x\|_1} \leqslant \frac{2\|x - y\|_1}{\|x\|_1} \qquad (7.14) \\
&\leqslant 2\eta\bar{\alpha} \qquad (7.15)
\end{aligned}$$

where we use the triangle inequality for (7.13), the reverse triangle inequality for the inequality in (7.14), and (7.15) follows from (7.12). If the noise level satisfies $\bar{\alpha} \leqslant \delta/2\eta$, then we know the iterates computed using the noisy and true gradients are at most δ away. □

Remark 7.4. *This result clearly shows the interplay between the noise level α and the step size η. When a sharp image f undergoes convolution followed by the addition of noise, Lemma 7.3 tells us that it is better to take short steps instead of being overtly aggressive.*

Why are short steps sufficient in practice? Given that every pixel in the blurred image is a nonnegative combination of neighboring pixels in the sharp image, it is enough to search among its *neighbors* to form a realistic image rather than searching over the whole image space. This can be performed efficiently using short steps.

7.4.2 Implementation Details

Initialization. We follow the standard practice common across many vision problems and estimate both f and k at many resolutions. More specifically, our estimation proceeds through a coarse-to-fine pyramid scheme. For each level, we run PRIDA (Algorithm 10) and upscale the resulting estimated image and kernel for the next level.

At the coarsest level, we initialize the kernel to be uniform, that is, $k_i^0 = 1/s$. While this choice of initialization is critical for many existing algorithms Perrone and

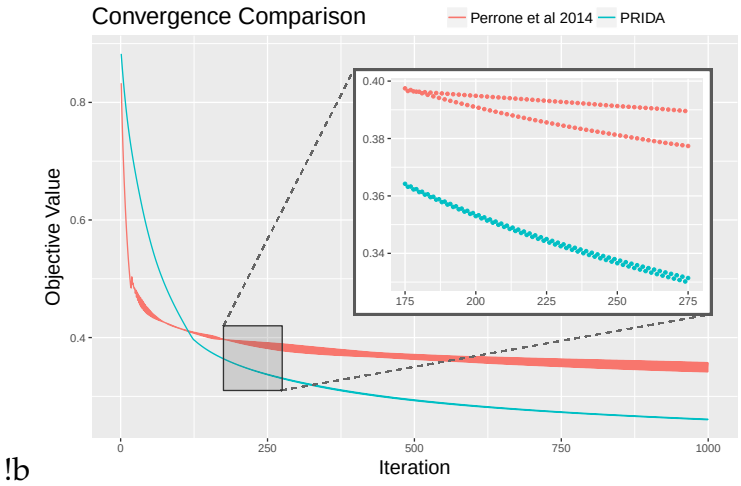


Figure 7.2: Convergence rate comparison of PRIDA (in red) and Perrone et al. in Perrone and Favaro (2014) (in blue).

Favaro (2014); Pan et al. (2014), it is not so important for PRIDA. Because the objective function is (jointly) bilinear, it may be the case that the initial few gradient steps will push some of the coordinates of the kernel to 0 after a Euclidean projection. This is problematic because it will remain at 0 during the entire course of optimization (at that scale), thus reducing the effective of the pyramid scheme. PRIDA on the other hand can be thought of as a version of "soft-removal": the multiplicative nature will naturally force all elements of any given kernel to remain strictly positive at all times, and hence a few "bad" steps will not necessarily hurt the overall performance.

Numerical Considerations. When calculating the step size per pixel η_{k_i}, it may be the case that a given point in the kernel has already been driven close to 0. In this case if the (noisy) gradient is negative, however small, the computed step $e^{-\eta_{k_i} g_{k_i}}$ may be $+\infty$ if the value at that point has fallen below machine precision. To avoid these issues, we apply a "Big M" correction Nemirovski (chapter 4) such that the step taken is the minimum of $\{\exp\{-\eta_i g_i\}, M\}$, where M is a large positive constant. Intuitively, a large M will allow PRIDA to take larger steps, thus encouraging faster convergence. We fix $M = 1000$ throughout our experiments.

7.5 Experiments

All experiments were conducted using MATLAB 2017a running on a 12-core Xeon E5-2620 @ 2.4 GHz machine with 64GB RAM. For all experiments on images of size

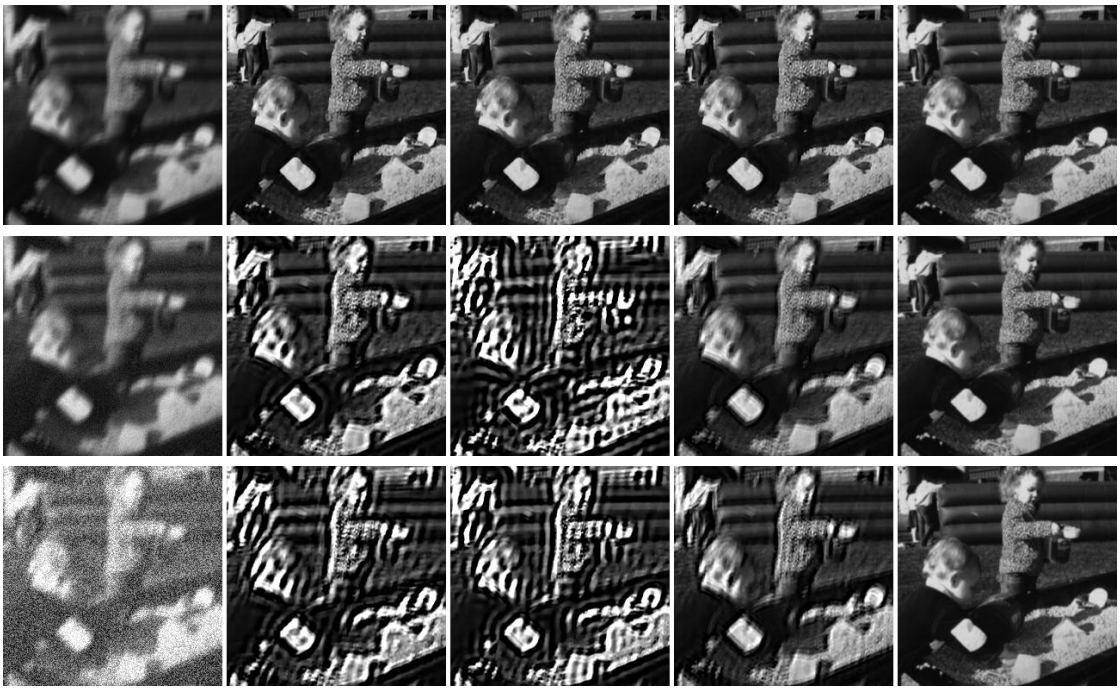


Figure 7.3: From left to right: (a) Input blurred images with added Gaussian noise. (b) Result from Perrone and Favaro (2014). (c) Result from Perrone et al. (2015). (d) Our Result. (e) Ground Truth. From top to bottom, each row corresponds to added noise with standard deviation 0,0.1, and 0.5 respectively.

255×255, we use a fixed regularization hyperparameter of $\lambda = 6e^{-4}$. The run time of each image on the finest scale is approximately 2-3 minutes. In the first two sets of experiments, our goal is to validate the theoretical properties of PRIDA shown in earlier sections viz., convergence and robustness. Finally, we test if PRIDA is efficient on real world color images. We compare with two recent standard baselines that are closely related to our algorithm Perrone and Favaro (2014); Perrone et al. (2015), and provide experimental details and comparisons with other algorithms.

7.5.1 Convergence

Figure 7.2 shows the function value convergence rates for PRIDA and for Perrone and Favaro (2014). Using the same pyramid scheme, we compute the function value for 1000 iterations of both algorithms over the finest level, fixing λ as stated above for PRIDA and the default setting provided by the authors in Perrone and Favaro (2014). Notice that while Perrone and Favaro (2014)'s method initially drops quickly,

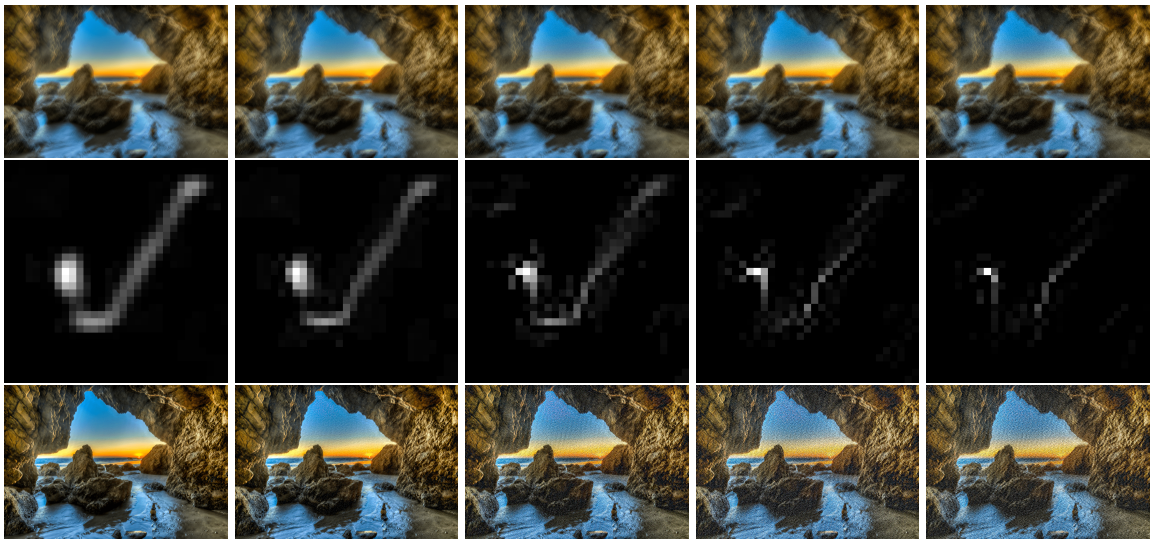


Figure 7.4: Color image recovery in the presence of intensity noise. From left to right, we add 0-mean Gaussian random noise with variance $0, 1, 4, 9, 16$ respectively. The first row shows the blurred and noisy input, the second the recovered kernel, and the third our final image recovery. Standard denoising methods can be applied to the deblurred image.

	Noise Level σ				
Alg.	1	3	5	7	9
Perrone and Favaro (2014)	0.0008	0.0223	0.0584	0.0849	0.0957
Perrone et al. (2015)	0.0006	0.0994	0.1375	0.1212	0.0941
PRIDA	0.0008	0.0041	0.0089	0.0149	0.0231

Table 7.1: Average endpoint error across the dataset in Levin et al. (2009) for varying levels of noise.

our method eventually converges much faster to a lower objective function. We note also that the PRIDA updates are significantly more stable, providing evidence of our robustness analysis above.

7.5.2 Robustness

To exemplify the robustness of PRIDA to noise, we conduct experiments on the well-known dataset first introduced by Levin et al. (2009). The grayscale images are 255×255 pixels in size with known blur kernels ranging in size from 13 to 27 pixels square. To evaluate robustness, we add varying levels of noise to each image, and

qualitatively evaluate the end result. We compare our method to the algorithms presented in Perrone and Favaro (2014) and in Perrone et al. (2015).

We show the results of PRIDA in comparison with the standard baselines in Figure 7.3. To generate the noisy and blurred images, Gaussian random noise with mean $\mu = 0$ was added to each blurred image. Here we can clearly observe the ability of our procedure to handle large amounts of noise. Over the entire dataset, we observe that in some interesting cases both algorithms from Perrone and Favaro (2014) and Perrone et al. (2015) are able to recover a reasonably sharp image in the presence of noise. Over the entire dataset from Levin et al. (2009), however, we note that their results are significantly more variable than that of PRIDA. On average, PRIDA is much more consistent in recovery over the entire dataset, shown in Table 7.1, validating our theoretical analysis above.

7.5.3 Scaling Up to the Real World

While the results above are valuable in validating our theoretical claims, we also evaluate our algorithms' robustness on real world images. Computationally, an interesting property of PRIDA is that all of its operations involve convolutions (Fast Fourier Transforms) and elementwise operations, both of which can benefit from GPU efficiencies.

We apply PRIDA to a set of large, color images that have been synthetically blurred. A comparative study on modern blind deconvolution algorithms compiled a dataset of synthetically-blurred spanning a wide range of image sizes, image content, and blur difficulty Lai et al. (2016). 25 real-world images collected from the Internet were each uniformly blurred with 4 known kernels of various size and support. Applying our algorithm to these images we find results comparable to state-of-the-art. In order to find an appropriate regularization, we perform a mild parameter sweep across all 25 images simultaneously for a given kernel size. For a kernel size of 31×31, we find that $\lambda = 2e^{-4}$ leads to the best qualitative results. Results on the front page include samples from this set.

To demonstrate the robustness of PRIDA on color images, noise was added to each pixel's lightness value in LAB space Wyszecki and Stiles (1982) and converted back to the original RGB color space. Figure 7.4 shows how our recovery is affected by increasing amounts of Gaussian random noise. While our kernel recovery degrades with more added noise, it is clear that we are still able to recover the kernel structure,

and that our final recovered image is in fact deblurred. Here, we present the raw output of our proposed model. Since the literature on denoising algorithms is mature, if necessary, a denoising algorithm can easily be run after PRIDA to remove the noise depending on its type. In fact, it is a common practice to have a "non-blind" stage at the end of the fine scale in many existing deblurring algorithms.

7.6 Conclusion

We propose a new algorithm, PRIDA, for recovering sharp images through blind deconvolution. PRIDA uniquely takes advantage of the specific problem domain, employing mirror descent over the simplex constraint set. We present theoretical analysis of PRIDA and derive guarantees on both convergence and robustness with no extra assumptions. In most real world settings, as noted by Zhu and Milanfar (2011), low light conditions and auto-focus software systems may introduce extra blur and noise since they depend on both exposure time and camera settings. Our exhaustive experimentation shows that PRIDA can be a comprehensive solution for real world problems. We showed both qualitatively and quantitively that PRIDA performs as good as the state of the art under no noise conditions and unarguably better in the presence of noise. We believe that our results will be a strong foundation not only for single image blind deconvolution problems, but also for furthering the success of recent data driven approaches such as deep learning architectures. Our code and additional experiments can be accessed through our Github repository here.

Chapter 8

Conclusions

Until now, we have presented evidence to support our Hypothesis 1 using various theoretical and/or practical frameworks. In Chapter 3, we showed that constraints in the filter flow model can be imposed using parallel convex optimization tools, reducing the run time by orders of magnitude (in some cases). In Chapter 4, we showed how to construct coresets (of a dataset) when the loss function is nonsmooth. Specifically, we showed that it is possible to optimize nonsmooth loss functions using a sparsity preserving deterministic CG method in Algorithm 6. In Chapter 5, we showed that commonly used norms that are used to measure the complexity of DNN can be explicitly imposed during training. Using update procedures (in Section 5.4) that are informed by the constraints can be used to improve the accuracy of state of the art models used in vision pipelines. In Chapter 6, we proposed two formulations for a nonlinear combinatorial optimization problem arising in the context of designing experiments for sparse linear models. For our incoherence based formulation, we showed how we can round the fractional to get integer solutions without much loss of accuracy. In Chapter 7, we developed a mirror descent algorithm for the blind deconvolution problem with probability simplex constraints. We analyzed the convergence and robustness properties of our method in Algorithm 10 theoretically and using popular vision datasets. In this final chapter, we will point out future directions for the main concepts used in each of the chapters.

8.1 Learning Locality Sensitive Filter Flow Maps

The filter flow algorithm proposed in Algorithm 5 exploits the locality of the transformation of images to develop efficient optimization strategies. Even though, the proposed algorithm comes with theoretical guarantees, the algorithm is still quite slow to be adopted in real time applications, especially when the image contains millions of pixels. For standard datasets, it turns out that a multigrid deep network based model can be used to learn filter flow transformations between pairs of images as in (Kong and Fowlkes, 2019). Here, *multigrid* corresponds to the pyramid approach described in Sections 7.4.2, and 3.6. Often models are shared across grid/scale (assumed to be similar) to reduce model complexity. However, this can be undesirable especially when there is large variations between motion/depth across video sequences. In such cases, (image) locality sensitive update rule as in Algorithm 5 can lead to faster optimization schemes. Next, Chapter 3 shows that convex formulations can be used to estimate filter flow maps that perform well, empirically. This suggests that using filter flow type losses to learn the maps in an end-to-end differentiable learning framework can speed-up the estimation procedure. To accomplish this, we recall basic results regarding Ordinary Differential Equations (ODE). Under technical conditions on the function to be estimated, the Lax-Milgram theorem (Johnson, 2012) ensures that variational/weak formulation and the strong formulation of an ODE are equivalent. In our context, we can use the strong formulation of the filter flow problem to construct loss functions (that depend on the solutions of the strong formulation). To train the model parameters efficiently, it will be interesting to see if the ideas from (Chen et al., 2018) can provide us guidance.

8.2 Coreset based Sampling and Applications

Recall that as described in Chapter 4, a coreset is defined as a subset of a set which *approximates* the whole set. Recent works suggest that sampling in high dimensions can be accelerated using coresets (Campbell and Broderick, 2017; Bach et al., 2012b). Note that the techniques that we used to analyze our Algorithm 6 are quite general. Specifically, it would be interesting to explore the (sparsity) benefits provided by our Algorithm 6 to Bayesian inference problems in high dimensional settings. From the computer vision perspective, coresets can be extremely useful for few shot learning (Wang and Yao, 2019). In few shot learning, the goal is to train a classifier that

requires the least amount of samples. More specifically, in few shot learning settings, the number of classes can grow at a much faster rate than the number of samples. In most common *metric learning* based approaches, each class is often identified with a center/prototype which is used for classification. The center for a particular class is simply the mean of the training data that belong to that class, embedded on a representation space computed by a neural network. Essentially, it will be interesting to see if such a representation can easily be computed using a coreset instead of the full training dataset. For this, connections between coresets and optimal transport theory (as showed in (Claici and Solomon, 2018)) can help us design algorithms that implicitly compute such a coreset during training nonlinear models. Another approach for few shot learning is to pool individual/sub datasets (Triantafillou et al., 2019) to obtain a metadataset. A basic requirement in pooling is that the margin (distance) distribution induced by sub datasets be preserved while training on the pooled dataset. Note that enforcing such margin distribution constraints naively may not be practical since the number of decision variables scale linearly with the size of the individual datasets. Using the coreset of sub datasets can be beneficial to efficiently train the pooled dataset, when the size of the coreset scales sublinearly.

8.3 Explicitly Imposing Probabilistic Constraints

In Chapter 5, we showed that we can impose norm based constraints on the parameters of DNN while training. Recall that these norm based constraints are intended to measure the complexity of ML models. However, in many real world scenarios, such constraints may be too pessimistic or unnatural. In such cases, it is often the case that we require constraints on the predictions of our ML models. Designing methods to impose probabilistic constraints efficiently can be practically critical in some cases. For example, when using ML models in criminal justice contexts, we may want our models to avoid discrimination (Kilbertus et al., 2017). In such cases, we would like to learn models that satisfy moment based *fairness* constraints (Donini et al., 2018; Agarwal et al., 2018). Specifically, developing methods that can explicitly impose such moment constraints (and beyond) while training DNNs with provable convergence properties is an important open problem. For instance, consider the vision task of (single) object image classification. For this task, cross entropy loss is the most commonly used loss function. However, cross entropy loss is (or most

commonly used loss functions such as quadratic, hinge are) agnostic to the size of the object in an image. So assuming that we have access to the size of the object in each image, we may want the error rate to be balanced with respect to the sizes of the object present. Similarly, in the binary classification case, one may want to estimate models with high F_1 score which can be formulated as a probabilistic constraint (Cotter et al., 2018). Designing fast algorithms to impose probabilistic constraints and toolboxes so that these methods can be used off-the-shelf can be practically useful.

8.4 Faster Finetuning via Experimental Design

In Chapter 6, we proposed formulations for an extension of the classical linear experimental design problem. A natural extension of the algorithms described in Chapter 6 to nonlinear f is interesting for various reasons. For example, say that we are to fine tune a DNN using SGD. In this case, the usual strategy is to obtain a uniformly random sample from the training set to estimate the gradient. A better option would be to use the local landscape properties of the loss function (Li et al., 2018; Fazlyab et al., 2019) since we are already close to a minima. Now, the experimental design problem in this context is to pick a subset of fixed cardinality (interpreted as the batch size) according to a criterion derived using the landscape properties. Then, we will update our model parameters using the gradient computed using this subset. Another direction to investigate is to extend our formulations to include more than two time points in the longitudinal study. In this setting, we would like to impose *regret* constraints on the enrollment trajectories of each individual involved in the study while maintaining the efficiency the algorithms. In particular, in clinical trials that span several years, an interesting research direction is to study the benefits of imposing constraints to improve the reliability of the study. For example, constraints of the form "if sample i is chosen at time point t, then i must be chosen from then on" tries to minimize the longitudinal stochastic error while estimating the asymptotic model parameters $\lim_{t\to\infty} \beta_t$ (in equation (6.2)). Designing efficient methods with convergence guarantees to impose such constraints explicitly is an open problem. However, note that it is possible that such constraints introduce bias while estimating the parameters of our model, for instance, an informative sample at time point t may be less informative at $t+1$, decreasing the *statistical* rate of estimation. From the statistical perspective, it will be interesting to quantify the asymptotic bias introduced

due to imposing such constraints.

8.5 Robustly Learning Kernels for Blind Deconvolution

In Chapter 7, we studied the blind deconvolution problem and developed the PRIDA Algorithm in Section 7.3. Recall that Algorithm 10 is based on the principle that geometry of the constraint set plays an important role in convergence and robustness properties of the algorithm and estimation properties of the parameters. (Xu et al., 2014) showed that it is possible to use deep networks to learn solutions of blind deconvolution problems. One approach is to learn parameters of the network by extracting *random* patches from the training images individually. Given the set of patches from a new test image, the network outputs the sharp estimate for each patch which are then combined appropriately to output the full sharp image. Unfortunately, the random patches based approach requires a lot of training samples and often takes long time to train. It may be possible to mitigate this problem by learning the blur kernel using a neural network, instead. Once a blur kernel is learned, the sharp image can simply be obtained by adapting our mirror descent algorithm as in Chapter 7. In particular, this is possible by assuming optimal blur kernels to be a convex combination of a dictionary of blur kernels. Note that in Chapter 7, we used the set of kernels with one nonzero entry as our dictionary. Secondly, it turns out that our discrete step gradient descent based optimization approach has a natural continuous time probabilistic interpretation. It may be possible to utilize ideas optimal transport theory (Lin et al., 2019) to develop efficient optimization schemes, and may provide a theoretically rigorous framework to learn blind deconvolution problems provably.

References

Abadi, Martín, et al. 2016. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv:1603.04467*.

Acharya, J., C. Daskalakis, and G. C. Kamath. 2015. Optimal testing for properties of distributions. In *Advances in neural information processing systems*.

Adams, Kenneth R. 1972. Perspective and the viewpoint. *Leonardo* 5(3):209–217.

Agarwal, Alekh, Alina Beygelzimer, Miroslav Dudik, John Langford, and Hanna Wallach. 2018. A reductions approach to fair classification. In *International conference on machine learning*, 60–69.

Agarwal, P. K., S. Har-Peled, and K. R. Varadarajan. 2005. Geometric approximation via coresets. *Combinatorial and computational geometry*.

Ageev, Alexander A, and Maxim I Sviridenko. 2004. Pipage rounding: A new method of constructing algorithms with proven performance guarantee. *Journal of Combinatorial Optimization* 8(3):307–328.

Alizadeh, Farid. 1995. Interior point methods in semidefinite programming with applications to combinatorial optimization. *SIAM journal on Optimization* 5(1):13–51.

Alpaydin, Ethem. 2009. *Introduction to machine learning*.

Amos, Brandon. 2016. Image Completion with Deep Learning in TensorFlow. http://bamos.github.io/2016/08/09/deep-completion. Accessed: [09/05/2018].

Anand, Rimmi, Divya Aggarwal, and Vijay Kumar. 2017. A comparative analysis of optimization solvers. *Journal of Statistics and Management Systems* 20(4):623–635.

Andreopoulos, Alexander, and John K Tsotsos. 2013. 50 years of object recognition: Directions forward. *Computer vision and image understanding* 117(8):827–891.

Anstreicher, Kurt M. 1996. On long step path following and sumt for linear and quadratic programming. *SIAM Journal on Optimization* 6(1):33–46.

Argyriou, A., M. Signoretto, and J Suykens. 2014. Hybrid conditional gradient-smoothing algorithms with applications to sparse and low rank regularization. *Regularization, Optimization, Kernels, and Support Vector Machines*.

Arora, Sanjeev, et al. 2017. Generalization and equilibrium in generative adversarial nets (gans). In *Icml*.

Bach, F., R. Jenatton, J. Mairal, and G. Obozinski. 2012a. Optimization with sparsity-inducing penalties. *Foundations & Trends in Machine Learning*.

Bach, Francis, Simon Lacoste-Julien, and Guillaume Obozinski. 2012b. On the equivalence between herding and conditional gradient algorithms. In *Proceedings of the 29th international coference on international conference on machine learning*, 1355–1362. Omnipress.

Bachem, Olivier, Mario Lucic, and Silvio Lattanzi. 2018. One-shot coresets: The case of k-clustering. In *International conference on artificial intelligence and statistics*.

Baker, Simon, Daniel Scharstein, JP Lewis, Stefan Roth, Michael J Black, and Richard Szeliski. 2011. A database and evaluation methodology for optical flow. *International Journal of Computer Vision* 92(1):1–31.

Balcan, M-F., S. Ehrlich, and Y. Liang. 2013. Distributed k-means and k-median clustering on general topologies. In *Advances in neural information processing systems*.

Baldi, Pierre, Peter Sadowski, and Daniel Whiteson. 2014. Searching for exotic particles in high-energy physics with deep learning. *Nature communications* 5:4308.

Ball, Nicholas M, and Robert J Brunner. 2010. Data mining and machine learning in astronomy. *International Journal of Modern Physics D* 19(07):1049–1106.

Bandeira, Afonso S, Edgar Dobriban, Dustin G Mixon, and William F Sawin. 2012. Certifying the restricted isometry property is hard. *arXiv preprint arXiv:1204.1580*.

Banerjee, O., L.E. Ghaoui, A. d'Aspremont, and G. Natsoulis. 2006. Convex optimization techniques for fitting sparse gaussian graphical models. In *Proceedings of the international conference on machine learning (icml)*.

Baykal, Cenk, Lucas Liebenwein, Igor Gilitschenski, Dan Feldman, and Daniela Rus. 2018. Data-dependent coresets for compressing neural networks with applications to generalization bounds. *arXiv preprint arXiv:1804.05345*.

Bengio, Yoshua. 2012. Practical recommendations for gradient-based training of deep architectures. In *Neural networks: Tricks of the trade*.

Bennett, K. P., and E. J. Bredensteiner. 2000. Duality and geometry in SVM classifiers. In *Proceedings of the international conference on machine learning (icml)*.

Berger, James O, and Thomas Sellke. 1987. Testing a point null hypothesis: the irreconcilability of p values and evidence. *Journal of the American statistical Association* 82(397):112–122.

Bertsekas, Dimitri P. Incremental gradient, subgradient, and proximal methods for convex optimization: A survey. *Optimization for Machine Learning* 2010:1–38.

Bertsekas, Dimitri P, and John N Tsitsiklis. 1989. *Parallel and distributed computation: numerical methods*. Prentice-Hall, Inc.

———. 2003. Parallel and distributed computation: numerical methods.

Bertsimas, Dimitris, Mac Johnson, and Nathan Kallus. 2015. The power of optimization over randomization in designing experiments involving small samples. *Operations Research*.

Bonesky, Thomas, Kristian Bredies, Dirk A Lorenz, and Peter Maass. 2007. A generalized conditional gradient method for nonlinear operator equations with sparsity constraints. *Inverse Problems* 23(5):2041.

Boyd, S, and L Vandenberghe. 2004a. *Convex optimization*.

Boyd, Stephen, and Lieven Vandenberghe. 2004b. *Convex optimization*. New York, NY, USA: Cambridge University Press.

Braverman, V., and S. R. Chestnut. 2014. Streaming sums in sublinear space. *CoRR*.

Bredies, Kristian, Dirk A Lorenz, and Peter Maass. 2009. A generalized conditional gradient method and its connection to an iterative shrinkage method. *Computational Optimization and Applications* 42(2):173–193.

Bubeck, Sébastien, et al. 2015. Convex optimization: Algorithms and complexity. *Foundations and Trends® in Machine Learning* 8(3-4):231–357.

Butler, D. J., J. Wulff, G. B. Stanley, and M. J. Black. 2012. A naturalistic open source movie for optical flow evaluation. In *European conf. on computer vision (eccv)*, ed. A. Fitzgibbon et al. (Eds.), 611–625. Part IV, LNCS 7577, Springer-Verlag.

Campbell, Trevor, and Tamara Broderick. 2017. Automated scalable bayesian inference via hilbert coresets. *arXiv preprint arXiv:1710.05053*.

Campisi, Patrizio, and Karen Egiazarian. 2016. *Blind image deconvolution: theory and applications*. CRC press.

Candès, Emmanuel J, and Yaniv Plan. 2009. Near-ideal model selection by l1 minimization. *The Annals of Statistics* 37(5A):2145–2177.

Candes, Emmanuel J, and Terence Tao. 2005. Decoding by linear programming. *Information Theory, IEEE Transactions on* 51(12):4203–4215.

Carli, Francesca P, Lipeng Ning, and Tryphon T Georgiou. 2013. Convex clustering via optimal mass transport. *arXiv preprint arXiv:1307.5459*.

Chakrabarti, Ayan. 2016. A neural approach to blind motion deblurring. In *European conference on computer vision*, 221–235. Springer.

Chaloner, Kathryn, and Isabella Verdinelli. 1995. Bayesian experimental design: A review. *Statistical Science* 273–304.

Chambolle, Antonin, and Pierre-Louis Lions. 1997. Image recovery via total variation minimization and related problems. *Numerische Mathematik* 76(2):167–188.

Chan, Timothy M. 2018. Applications of chebyshev polynomials to low-dimensional computational geometry. *Journal of Computational Geometry*.

Chan, Tony F, and Chiu-Kwong Wong. 1998. Total variation blind deconvolution. *IEEE transactions on Image Processing* 7(3):370–375.

Chekuri, Chandra, Jan Vondrák, and Rico Zenklusen. 2009. Dependent randomized rounding for matroid polytopes and applications. *arXiv preprint arXiv:0909.4348*.

Chen, Tian Qi, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. 2018. Neural ordinary differential equations. In *Advances in neural information processing systems*, 6571–6583.

Chen, Yudong, Srinadh Bhojanapalli, Sujay Sanghavi, and Rachel Ward. 2014. Coherent matrix completion. In *Proceedings of the 31st international conference on machine learning*, 674–682.

Chen, Yunmei, and Xiaojing Ye. 2011. Projection onto a simplex. *arXiv preprint arXiv:1101.6081*.

Cheng, Yu, et al. 2017. A survey of model compression and acceleration for deep neural networks. *arXiv:1710.09282*.

Cheung, Edward, and Yuying Li. 2017. Nonsmooth frank-wolfe using uniform affine approximations. *arXiv preprint arXiv:1710.05776*.

Cho, Sunghyun, and Seungyong Lee. 2009. Fast motion deblurring. In *Acm transactions on graphics (tog)*, vol. 28, 145. ACM.

Claici, Sebastian, and Justin Solomon. 2018. Wasserstein coresets for lipschitz costs. *arXiv preprint arXiv:1805.07412*.

Clarkson, K. L. 2008. Coresets, sparse greedy approximation, and the Frank-Wolfe algorithm. In *Proceedings of the symposium on discrete algorithms (soda)*.

Clarkson, K. L., and D. P. Woodruff. 2015. Input sparsity and hardness for robust subspace approximation. In *Foundations of computer science, FOCS*.

Clarkson, Kenneth L, Elad Hazan, and David P Woodruff. 2012. Sublinear optimization for machine learning. *Journal of the ACM (JACM)* 59(5):23.

Cohen, Michael B, Yin Tat Lee, Gary Miller, Jakub Pachocki, and Aaron Sidford. 2016. Geometric median in nearly linear time. In *Proceedings of the forty-eighth annual acm symposium on theory of computing*, 9–21. ACM.

Collins, Maxwell D, Ji Liu, Jia Xu, Lopamudra Mukherjee, and Vikas Singh. 2014. Spectral clustering with a convex regularizer on millions of images. In *Computer vision–eccv 2014*, 282–298. Springer.

Cotter, Andrew, Heinrich Jiang, Serena Wang, Taman Narayan, Maya Gupta, Seungil You, and Karthik Sridharan. 2018. Optimization with non-differentiable constraints with applications to fairness, recall, churn, and other goals. *arXiv preprint arXiv:1809.04198*.

Călinescu, G., H. Karloff, and Y. Rabani. 1998. An improved approximation algorithm for multiway cut. In *Proceedings of the symposium on theory of computing (stoc)*.

Cui, L. 2008. Maintenance models and optimization. In *Handbook of performability engineering*.

Dalal, Navneet, and Bill Triggs. 2005. Histograms of oriented gradients for human detection.

Das, Ashish. An introduction to optimality criteria and some results on optimal block design.

Daskalakis, Constantinos, Gautam Kamath, and John Wright. 2018. Which distribution distances are sublinearly testable? In *Proceedings of the twenty-ninth annual acm-siam symposium on discrete algorithms*. SIAM.

Dauphin, Yann, Harm de Vries, and Yoshua Bengio. 2015. Equilibrated adaptive learning rates for non-convex optimization. In *Nips*.

Deng, Jia, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *2009 ieee conference on computer vision and pattern recognition*, 248–255. Ieee.

Dette, Holger, Andrej Pepelyšev, and Anatolij A Žigljavskij. 2011. Optimal design for linear models with correlated observations.

Donini, Michele, Luca Oneto, Shai Ben-David, John S Shawe-Taylor, and Massimiliano Pontil. 2018. Empirical risk minimization under fairness constraints. In *Advances in neural information processing systems*, 2791–2801.

Drineas, P., R. Kannan, and M. W. Mahoney. 2006. Fast monte carlo algorithms for matrices i: Approximating matrix multiplication. *SIAM Journal on Computing*.

Drineas, Petros, Malik Magdon-Ismail, Michael W Mahoney, and David P Woodruff. 2012. Fast approximation of matrix coherence and statistical leverage. *The Journal of Machine Learning Research* 13(1):3475–3506.

Du, Hao, Dan B Goldman, and Steven M Seitz. 2011. Binocular photometric stereo. In *Bmvc*, 1–11. Citeseer.

Duan, Kun, Devi Parikh, David Crandall, and Kristen Grauman. 2012. Discovering localized attributes for fine-grained recognition. In *2012 ieee conference on computer vision and pattern recognition*, 3474–3481. IEEE.

Duchi, John, et al. 2008. Efficient projections onto the l 1-ball for learning in high dimensions. In *Icml*.

Efron, Bradley, Trevor Hastie, Iain Johnstone, Robert Tibshirani, et al. 2004. Least angle regression. *The Annals of statistics* 32(2):407–499.

Fadili, Jalal M, and Gabriel Peyré. 2011. Total variation projection with first order schemes. *IEEE Transactions on Image Processing*.

Fazlyab, Mahyar, Alexander Robey, Hamed Hassani, Manfred Morari, and George J. Pappas. 2019. Efficient and accurate estimation of lipschitz constants for deep neural networks. 1906.04893.

Feldman, D., M. Monemizadeh, C. Sohler, and D. P. Woodruff. 2010. Coresets and sketches for high dimensional subspace approximation problems. In *Proceedings of the symposium on discrete algorithms (soda)*.

Feldman, D., A. Sugaya, and D. Rus. 2012. An effective coreset compression algorithm for large scale sensor networks. In *Proceedings of the international conference on information processing in sensor networks (ipsn)*.

Ferguson, Allen R, and George B Dantzig. 1956. The allocation of aircraft to routesâĽ"an example of linear programming under uncertain demand. *Management science* 3(1):45–73.

Ferris, Michael C, Olvi L Mangasarian, and Stephen J Wright. 2007. *Linear programming with matlab*, vol. 7. SIAM.

Fleet, David J, Michael J Black, Yaser Yacoob, and Allan D Jepson. 2000. Design and use of linear models for image motion analysis. *International Journal of Computer Vision* 36(3):171–193.

Frank, M., and P. Wolfe. 1956. An algorithm for quadratic programming. *Naval Research Logistics Quarterly*.

Frerix, Thomas, Daniel Cremers, and Matthias Nießner. 2019. Linear inequality constraints for neural network activations. *arXiv preprint arXiv:1902.01785*.

Frerix, Thomas, et al. 2017. Proximal backpropagation. *arXiv:1706.04638*.

Freund, Robert M, and Paul Grigas. 2013. New analysis and results for the conditional gradient method.

Garber, D., and E. Hazan. 2011. Approximating semidefinite programs in sublinear time. In *Advances in neural information processing systems*.

Garber, D., and O. Meshi. 2016. Linear-memory and Decomposition-invariant Linearly Convergent Conditional Gradient Algorithm for Structured Polytopes. *ArXiv e-prints*.

Garber, Dan, and Elad Hazan. 2015. Faster rates for the frank-wolfe method over strongly-convex sets. In *32nd international conference on machine learning, icml 2015*.

Garber, Dan, and Ofer Meshi. 2016. Linear-memory and decomposition-invariant linearly convergent conditional gradient algorithm for structured polytopes. In *Advances in neural information processing systems*, 1001–1009.

Gärtner, B., and M. Jaggi. 2009. Coresets for polytope distance. In *Proceedings of the acm symposium on computation geometry (socg)*.

Gearhart, Jared Lee, Kristin Lynn Adair, Justin David Durfee, Katherine A Jones, Nathaniel Martin, and Richard Joseph Detry. 2013. Comparison of open-source linear programming solvers. Tech. Rep., Sandia National Lab.(SNL-NM), Albuquerque, NM (United States).

Ghadimi, Saeed, and Guanghui Lan. 2016. Accelerated gradient methods for nonconvex nonlinear and stochastic programming. *Mathematical Programming* 156(1-2): 59–99.

Gidel, G., T. Jebara, and S. Lacoste-Julien. 2017. Frank-wolfe algorithms for saddle point problems. In *Proceedings of the artificial intelligence and statistics (aistats)*.

Gilbert, E. G. 1966. An iterative procedure for computing the minimum of a quadratic form on a convex set. *SIAM Journal on Control*.

Goemans, M, and D P Williamson. 1995. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *JACM*.

Golub, Gene H, and Charles F Van Loan. 2012. *Matrix computations*.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. 2016. *Deep learning*.

Gorski, Jochen, Frank Pfeuffer, and Kathrin Klamroth. 2007. Biconvex sets and optimization with biconvex functions: a survey and extensions. *Mathematical Methods of Operations Research* 66(3):373–407.

Grant, Michael, and Stephen Boyd. 2014. CVX: Matlab software for disciplined convex programming, version 2.1. http://cvxr.com/cvx.

Grbovic, M., C. R. Dance, and S. Vucetic. 2012. Sparse principal component analysis with constraints. In *Proceedings of the aaai conference on artificial intelligence*.

Günlük, O., and J. Linderoth. 2012. Perspective reformulation and applications. In *Mixed integer nonlinear programming*.

Hadfield-Menell, Dylan, Christopher Lin, Rohan Chitnis, Stuart Russell, and Pieter Abbeel. 2016. Sequential quadratic programming for task plan optimization. In *2016 ieee/rsj international conference on intelligent robots and systems (iros)*, 5040–5047. IEEE.

Hamilton, Alan G. 1988. *Logic for mathematicians*. Cambridge University Press.

Har-Peled, S., and A. Kushal. 2005. Smaller coresets for k-median and k-means clustering. In *Proceedings of the acm symposium on computation geometry (socg)*.

Har-Peled, S., D. Roth, and D. Zimak. 2007. Maximum margin coresets for active and noise tolerant learning. In *International joint conference on artifical intelligence*.

Har-Peled, Sariel, and Akash Kushal. 2007. Smaller coresets for k-median and k-means clustering. *Discrete & Computational Geometry* 37(1):3–19.

Harchaoui, Zaid, Anatoli Juditsky, and Arkadi Nemirovski. 2014. Conditional gradient algorithms for norm-regularized smooth convex optimization. *Mathematical Programming* 1–38.

Hardt, Moritz. 2014. Understanding alternating minimization for matrix completion. In *Foundations of computer science (focs), 2014 ieee 55th annual symposium on*, 651–660. IEEE.

Hardt, Moritz, Ben Recht, and Yoram Singer. 2016. Train faster, generalize better: Stability of stochastic gradient descent. In *Icml*.

Hardt, Moritz, Benjamin Recht, and Yoram Singer. 2015. Train faster, generalize better: Stability of stochastic gradient descent. *arXiv preprint arXiv:1509.01240*.

Hartley, Richard, and Andrew Zisserman. 2003. *Multiple view geometry in computer vision*. Cambridge university press.

Harvey, Nicholas JA, and Neil Olver. 2014. Pipage rounding, pessimistic estimators and matrix concentration. In *Proceedings of the twenty-fifth annual acm-siam symposium on discrete algorithms*, 926–945. SIAM.

Hastie, Trevor, Robert Tibshirani, and Jerome Friedman. 2001. *The elements of statistical learning*. Springer Series in Statistics, New York, NY, USA: Springer New York Inc.

Hazan, E., and S. Kale. 2012. Projection-free online learning. In *Proceedings of the international conference on machine learning (icml)*.

He, Kaiming, et al. 2016. Deep residual learning for image recognition. In *Cvpr*.

Heaton, JB, NG Polson, and Jan Hendrik Witte. 2017. Deep learning for finance: deep portfolios. *Applied Stochastic Models in Business and Industry* 33(1):3–12.

Heo, Yong Seok, Kyoung Mu Lee, and Sang Uk Lee. 2008. Illumination and camera invariant stereo matching. In *Computer vision and pattern recognition, 2008. cvpr 2008. ieee conference on*, 1–8. IEEE.

Hinrichs, Chris, N Maritza Dowling, Sterling C Johnson, and Vikas Singh. 2012. Mkl-based sample enrichment and customized outcomes enable smaller ad clinical trials. In *Machine learning and interpretation in neuroimaging*, 124–131. Springer.

Hinrichs, Chris, Vikas Singh, Guofan Xu, Sterling C Johnson, Alzheimers Disease Neuroimaging Initiative, et al. 2011. Predictive markers for ad in a multi-modality framework: an analysis of mci progression in the adni population. *Neuroimage* 55(2):574–589.

Hiriart-Urruty, J.-P., and C. Lemaréchal. 1993. *Convex analysis and minimization algorithms*. Springer.

Hirsch, Michael, Christian J Schuler, Stefan Harmeling, and Bernhard Schölkopf. 2011. Fast removal of non-uniform camera shake. In *Computer vision (iccv), 2011 ieee international conference on*, 463–470. IEEE.

Hirsch, Michael, Suvrit Sra, Bernhard Scholkopf, and Stefan Harmeling. 2010. Efficient filter flow for space-variant multiframe blind deconvolution.

Hirschmüller, Heiko, and Daniel Scharstein. 2009. Evaluation of stereo matching costs on images with radiometric differences. *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 31(9):1582–1599.

Hofmann, Thomas, Bernhard Schölkopf, and Alexander J Smola. 2008. Kernel methods in machine learning. *The annals of statistics* 1171–1220.

Horel, Thibaut, Stratis Ioannidis, and S Muthukrishnan. 2014. Budget feasible mechanisms for experimental design. In *Latin 2014: Theoretical informatics*, 719–730. Springer.

Horn, Berthold KP, and Brian G Schunck. 1981. Determining optical flow. *Artificial intelligence* 17(1-3):185–203.

Howard, Andrew G, et al. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv:1704.04861*.

Huggins, Jonathan, Trevor Campbell, and Tamara Broderick. 2016. Coresets for scalable bayesian logistic regression. In *Advances in neural information processing systems*.

Ithapu, Vamsi K, Vikas Singh, Ozioma C Okonkwo, Richard J Chappell, N Maritza Dowling, Sterling C Johnson, Alzheimer's Disease Neuroimaging Initiative, et al. 2015. Imaging-based enrichment criteria using deep learning algorithms for efficient clinical trials in mild cognitive impairment. *Alzheimer's & Dementia* 11(12):1489–1499.

Jaggi, M. 2011. Sparse convex optimization methods for machine learning. Ph.D. thesis, Eidgenössische Technische Hochschule ETH Zürich, Nr. 20013.

———. 2013. Revisiting Frank-Wolfe: Projection-free sparse convex optimization. In *Proceedings of the international conference on machine learning (icml)*.

Jain, Anil K, and Richard C Dubes. 1988. Algorithms for clustering data. *Englewood Cliffs: Prentice Hall, 1988*.

Jain, Prateek, and Abhradeep Thakurta. 2012. Mirror descent based database privacy. *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques* 579–590.

Jin, Meiguang, Zhe Hu, and Paolo Favaro. 2019. Learning to extract flawless slow motion from blurry videos. In *Proceedings of the ieee conference on computer vision and pattern recognition*, 8112–8121.

Jin, Meiguang, Stefan Roth, and Paolo Favaro. 2017. Noise-blind image deblurring. In *Computer vision and pattern recognition (cvpr), 2017 ieee conference on. ieee*.

———. 2018. Normalized blind deconvolution. In *Proceedings of the european conference on computer vision (eccv)*, 668–684.

Johnson, Claes. 2012. *Numerical solution of partial differential equations by the finite element method*. Courier Corporation.

Johnson, Rie, and Tong Zhang. 2013. Accelerating stochastic gradient descent using predictive variance reduction. In *Nips*.

Juditsky, Anatoli, and Arkadi Nemirovski. 2011. On verifiable sufficient conditions for sparse signal recovery via â„“ 1 minimization. *Mathematical programming* 127(1): 57–88.

Juditsky, Anatoli, Arkadi Nemirovski, et al. First order methods for nonsmooth convex large-scale optimization, i: general purpose methods. *Optimization for Machine Learning*.

Kanade, Takeo, and Masatoshi Okutomi. 1991. A stereo matching algorithm with an adaptive window: Theory and experiment. In *Proceedings. 1991 ieee international conference on robotics and automation*, 1088–1095. IEEE.

Kempthorne, Oscar. 1952. The design and analysis of experiments.

Kilbertus, Niki, Mateo Rojas Carulla, Giambattista Parascandolo, Moritz Hardt, Dominik Janzing, and Bernhard Schölkopf. 2017. Avoiding discrimination through causal reasoning. In *Advances in neural information processing systems*, 656–666.

Kingma, Diederik, and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kirk, Roger E. 1982. *Experimental design*. Wiley Online Library.

Kolmogorov, V. 2015. Commutativity in the random walk formulation of the lovasz local lemma. *arXiv preprint arXiv:1506.08547*.

Komiya, Hidetoshi. 1988. Elementary proof for sion's minimax theorem. *Kodai mathematical journal* 11(1):5–7.

Kong, Shu, and Charless Fowlkes. 2019. Multigrid predictive filter flow for unsupervised learning on videos. *arXiv preprint arXiv:1904.01693*.

Konstantinou, Maria, Stefanie Biedermann, and Alan Kimber. 2011. Optimal designs for two-parameter nonlinear models with application to survival models.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, 1097–1105.

Lacoste-Julien, Simon. 2016. Convergence rate of frank-wolfe for non-convex objectives. *arXiv:1607.00345*.

Lacoste-Julien, Simon, and Martin Jaggi. 2015. On the global linear convergence of frank-wolfe optimization variants. In *Nips*.

Lai, Wei-Sheng, Jia-Bin Huang, Zhe Hu, Narendra Ahuja, and Ming-Hsuan Yang. 2016. A comparative study for single image blind deblurring. In *Proceedings of the ieee conference on computer vision and pattern recognition*, 1701–1709.

Lan, G. 2013. The complexity of large-scale convex programming under a linear optimization oracle. *arXiv preprint arXiv:1309.5550*.

Lan, Guanghui, Arkadi Nemirovski, and Alexander Shapiro. 2012. Validation analysis of mirror descent stochastic approximation method. *Mathematical programming*.

Landau, SM, D Harvey, CM Madison, EM Reiman, NL Foster, PS Aisen, RC Petersen, LM Shaw, JQ Trojanowski, CR Jack, et al. 2010. Comparing predictors of conversion and decline in mild cognitive impairment. *Neurology* 75(3):230–238.

Lazebnik, Svetlana, Cordelia Schmid, and Jean Ponce. 2004. Semi-local affine parts for object recognition. In *British machine vision conference (bmvc'04)*, 779–788. The British Machine Vision Association (BMVA).

Le, Quoc V, Marc'Aurelio Ranzato, Rajat Monga, Matthieu Devin, Kai Chen, Greg S Corrado, Jeff Dean, and Andrew Y Ng. 2012. Building high-level features using large scale unsupervised learning. In *Proceedings of the 29th international coference on international conference on machine learning*, 507–514. Omnipress.

Lee, J. D., I. Panageas, G. Piliouras, M. Simchowitz, M. I. Jordan, and B. Recht. 2017. First-order Methods Almost Always Avoid Saddle Points. 1710.07406.

Lee, S., and S. J. Wright. 2012. Manifold identification in dual averaging for regularized stochastic online learning. *Journal of Machine Learning Research*.

Lee, Seung-Chul, and Yong-Hwan Kim. 2002. An enhanced lagrangian neural network for the eld problems with piecewise quadratic cost functions and nonlinear constraints. *Electric Power Systems Research*.

Levin, Anat, Yair Weiss, Fredo Durand, and William T Freeman. 2009. Understanding and evaluating blind deconvolution algorithms. In *Computer vision and pattern recognition, 2009. cvpr 2009. ieee conference on*, 1964–1971. IEEE.

Li, Chao, Yi Yang, Min Feng, Srimat Chakradhar, and Huiyang Zhou. 2016a. Optimizing memory efficiency for deep convolutional neural networks on gpus. In *Sc'16: Proceedings of the international conference for high performance computing, networking, storage and analysis*, 633–644. IEEE.

Li, Chen, Shuochen Su, Yasuyuki Matsushita, Kun Zhou, and Stephen Lin. 2013. Bayesian depth-from-defocus with shading constraints. In *Proceedings of the ieee conference on computer vision and pattern recognition*, 217–224.

Li, Hao, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. 2018. Visualizing the loss landscape of neural nets. In *Advances in neural information processing systems*, 6389–6399.

Li, Jia. 2008. Linear, ridge regression, and principal component analysis.

Li, Stan Z. 1994. Markov random field models in computer vision. In *European conference on computer vision*, 361–370. Springer.

Li, Xin, and Yuhong Guo. 2013. Adaptive active learning for image classification. In *Computer vision and pattern recognition (cvpr), 2013 ieee conference on*, 859–866. IEEE.

Li, Yanjun, Kiryung Lee, and Yoram Bresler. 2016b. Identifiability in blind deconvolution with subspace or sparsity constraints. *IEEE Transactions on Information Theory* 62(7):4266–4275.

Lin, Tianyi, Nhat Ho, and Michael I Jordan. 2019. On efficient optimal transport: An analysis of greedy and accelerated mirror descent algorithms. *arXiv preprint arXiv:1901.06482*.

Liu, H., M. Palatucci, and J. Zhang. 2009. Blockwise coordinate descent procedures for the multi-task lasso, with applications to neural semantic basis discovery.

Lowe, David G. 2004. Distinctive image features from scale-invariant keypoints. *International journal of computer vision* 60(2):91–110.

Luong, Duy VN, Panos Parpas, Daniel Rueckert, and Berç Rustem. 2012. Solving mrf minimization by mirror descent. In *International symposium on visual computing*, 587–598. Springer.

Mahadevan, Sridhar, and Bo Liu. 2012. Sparse q-learning with mirror descent. In *Proceedings of the twenty-eighth conference on uncertainty in artificial intelligence*, 564–573. AUAI Press.

Mangasarian, O. L. 1999. Generalized support vector machines. *Advances in Neural Information Processing Systems (NIPS)*.

Márquez-Neila, Pablo, Mathieu Salzmann, and Pascal Fua. 2017. Imposing hard constraints on deep networks: Promises and limitations. *arXiv:1706.02025*.

Mei, Xing, Xun Sun, Weiming Dong, Haitao Wang, and Xiaopeng Zhang. 2013. Segment-tree based cost aggregation for stereo matching. In *Proceedings of the ieee conference on computer vision and pattern recognition*, 313–320.

Menze, Moritz, Christian Heipke, and Andreas Geiger. 2015. Discrete optimization for optical flow. In *German conference on pattern recognition*, 16–28. Springer.

Michaeli, Tomer, and Michal Irani. 2014. Blind deblurring using internal patch recurrence. In *European conference on computer vision*, 783–798. Springer.

Midler, JL, and RD Wollmer. 1969. Stochastic programming models for scheduling airlift operations. *Naval Research Logistics Quarterly* 16(3):315–330.

Mikolov, Tomas, et al. 2014. Learning longer memory in recurrent neural networks. *arXiv:1412.7753*.

Moosavi Dezfooli, Seyed Mohsen, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. 2017. Universal adversarial perturbations. In *Proceedings of 2017 ieee conference on computer vision and pattern recognition (cvpr)*. EPFL-CONF-226156.

Moosavi-Dezfooli, Seyed-Mohsen, Alhussein Fawzi, and Pascal Frossard. 2016. Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the ieee conference on computer vision and pattern recognition*, 2574–2582.

Murty, Katta G, and Santosh N Kabadi. 1987. Some np-complete problems in quadratic and nonlinear programming. *Mathematical programming* 39(2):117–129.

Nah, Seungjun, Tae Hyun Kim, and Kyoung Mu Lee. 2016. Deep multi-scale convolutional neural network for dynamic scene deblurring. *arXiv preprint arXiv:1612.02177*.

Nathan, V., and S. Raghvendra. 2014. Accurate Streaming Support Vector Machines. *ArXiv e-prints*.

Nemirovski, Arkadi. Interior point polynomial time methods in convex programming.

———. 2012. Tutorial: Mirror descent algorithms for large-scale deterministic and stochastic convex optimization. *Edinburgh*.

Nesterov, Y. 1998. Introductory lectures on convex programming volume i: Basic course.

———. 2005. Smooth minimization of non-smooth functions. *Mathematical programming*.

———. 2009. Primal-dual subgradient methods for convex problems. Tech. Rep.

———. 2013. *Introductory lectures on convex optimization: A basic course*, vol. 87. Springer Science & Business Media.

———. 2015. Complexity bounds for primal-dual methods minimizing the model of objective function. Tech. Rep.

Netzer, Yuval, et al. Reading digits in natural images with unsupervised feature learning.

Newcombe, Richard A, Dieter Fox, and Steven M Seitz. 2015. Dynamicfusion: Reconstruction and tracking of non-rigid scenes in real-time. In *Proceedings of the ieee conference on computer vision and pattern recognition*, 343–352.

Neyshabur, Behnam, Ruslan R Salakhutdinov, and Nati Srebro. 2015a. Path-sgd: Path-normalized optimization in deep neural networks. In *Nips*.

Neyshabur, Behnam, Ryota Tomioka, and Nathan Srebro. 2015b. Norm-based capacity control in neural networks. In *Colt*.

Niu, F., B. Recht, C. Ré, and S. J. Wright. 2011. Hogwild!: A lock-free approach to parallelizing stochastic gradient descent. In *Neural information processing systems*.

Nocedal, J., and S. J. Wright. 2006. *Numerical optimization*. 2nd ed. New York: Springer.

Noroozi, Mehdi, Paramanand Chandramouli, and Paolo Favaro. 2017. Motion deblurring in the wild. *arXiv preprint arXiv:1701.01486*.

Obermeyer, Ziad, and Ezekiel J Emanuel. 2016. Predicting the futureâĽ"big data, machine learning, and clinical medicine. *The New England journal of medicine* 375(13): 1216.

Oktay, Ozan, et al. 2017. Anatomically constrained neural networks (acnn): Application to cardiac image enhancement and segmentation. *arXiv:1705.08302*.

Osher, Stanley, Martin Burger, Donald Goldfarb, Jinjun Xu, and Wotao Yin. 2005. An iterative regularization method for total variation-based image restoration. *Multiscale Modeling & Simulation* 4(2):460–489.

Pan, Jinshan, Zhe Hu, Zhixun Su, and Ming-Hsuan Yang. 2014. Deblurring text images via l0-regularized intensity and gradient prior. In *Proceedings of the ieee conference on computer vision and pattern recognition*, 2901–2908.

Panageas, Ioannis, and Georgios Piliouras. 2016. Gradient descent only converges to minimizers: Non-isolated critical points and invariant regions. *arXiv preprint arXiv:1605.00405*.

Panteleev, Jane, Hua Gao, and Lei Jia. 2018. Recent applications of machine learning in medicinal chemistry. *Bioorganic & medicinal chemistry letters* 28(17):2807–2815.

Parikh, N., and S. P. Boyd. 2014. Proximal algorithms. *Foundations and Trends in Optimization*.

Pathak, Deepak, Philipp Krahenbuhl, and Trevor Darrell. 2015. Constrained convolutional neural networks for weakly supervised segmentation. In *Iccv*.

Pennanen, T. 2012. Introduction to convex optimization in financial markets. *Mathematical Programming*.

Perrone, Daniele, Remo Diethelm, and Paolo Favaro. 2015. Blind deconvolution via lower-bounded logarithmic image priors. In *International workshop on energy minimization methods in computer vision and pattern recognition*, 112–125. Springer.

Perrone, Daniele, and Paolo Favaro. 2014. Total variation blind deconvolution: The devil is in the details. In *Proceedings of the ieee conference on computer vision and pattern recognition*, 2909–2916.

———. 2016. A clearer picture of total variation blind deconvolution. *IEEE transactions on pattern analysis and machine intelligence* 38(6):1041–1055.

Pesaran, M Hashem, and Melvyn Weeks. 2001. Non-nested hypothesis testing: an overview. *A Companion to Theoretical Econometrics* 279–309.

Pierucci, F., Z. Harchaoui, and J. Malick. 2014. A smoothing approach for composite conditional gradient with nonsmooth loss. Ph.D. thesis, INRIA Grenoble.

Pilanci, Mert, Laurent E Ghaoui, and Venkat Chandrasekaran. 2012. Recovery of sparse probability measures via convex programming. In *Advances in neural information processing systems*, 2420–2428.

Platt, John C, and Alan H Barr. 1988. Constrained differential optimization. In *Nips*.

Pukelsheim, Friedrich. 1993. *Optimal design of experiments*, vol. 50. siam.

Raayoni, Gal, George Pisha, Yahel Manor, Uri Mendlovic, Doron Haviv, Yaron Hadad, and Ido Kaminer. 2019. The ramanujan machine: Automatically generated conjectures on fundamental constants. *arXiv preprint arXiv:1907.00205*.

Raja, Yogesh, Stephen J McKenna, and Shaogang Gong. 1998. Segmentation and tracking using colour mixture models. In *Asian conference on computer vision*, 607–614. Springer.

Ramakrishnan, Sainandan, Shubham Pachori, Aalok Gangopadhyay, and Shanmuganathan Raman. 2017. Deep generative filter for motion deblurring. *arXiv preprint arXiv:1709.03481*.

Ravi, Sathya N, Maxwell D Collins, and Vikas Singh. 2019a. A deterministic nonsmooth frank wolfe algorithm with coreset guarantees. *Informs Journal on Optimization* ijoo–2019.

Ravi, Sathya N, Tuan Dinh, Vishnu Suresh Lokhande, and Vikas Singh. 2019b. Explicitly imposing constraints in deep networks via conditional gradients gives improved generalization and faster convergence. In *Proceedings of the aaai conference on artificial intelligence*, vol. 33, 4772–4779.

Ravi, Sathya N., Yunyang Xiong, Lopamudra Mukherjee, and Vikas Singh. 2017. Filter flow made practical: Massively parallel and lock-free. In *The ieee conference on computer vision and pattern recognition (cvpr)*.

Recht, Benjamin, Maryam Fazel, and Pablo A Parrilo. 2010. Guaranteed minimum-rank solutions of linear matrix equations via nuclear norm minimization. *SIAM review*.

Recht, Benjamin, Christopher Re, Stephen Wright, and Feng Niu. 2011. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in neural information processing systems*, 693–701.

Reddi, Sashank J., et al. 2016. Stochastic frank-wolfe methods for nonconvex optimization. In *54th annual allerton conference*.

Revaud, Jerome, Philippe Weinzaepfel, Zaid Harchaoui, and Cordelia Schmid. 2015. Epicflow: Edge-preserving interpolation of correspondences for optical flow. In *Proceedings of the ieee conference on computer vision and pattern recognition*, 1164–1172.

Rhemann, Christoph, Carsten Rother, Pushmeet Kohli, and Margrit Gelautz. 2010. A spatially varying psf-based prior for alpha matting. In *Computer vision and pattern recognition (cvpr), 2010 ieee conference on*, 2149–2156. IEEE.

Rissanen, Jorma. 1985. *Minimum description length principle*.

Robinson, S. M. 2012. Convexity in finite-dimensional spaces. Preprint.

Roosta-Khorasani, Farbod, and Michael W Mahoney. 2019. Sub-sampled newton methods. *Mathematical Programming* 174(1-2):293–326.

Rosasco, Lorenzo, Ernesto De Vito, Andrea Caponnetto, Michele Piana, and Alessandro Verri. 2004. Are loss functions all the same? *Neural Computation* 16(5):1063–1076.

Rudd, Keith, Gianluca Di Muro, and Silvia Ferrari. 2014. A constrained backpropagation approach for the adaptive solution of partial differential equations. *IEEE transactions on neural networks and learning systems*.

Rudelson, M., and R. Vershynin. 2007. Sampling from large matrices: An approach through geometric functional analysis. *Journal of the ACM (JACM)*.

Ruder, Sebastian. 2017. An overview of multi-task learning in deep neural networks. *arXiv:1706.05098*.

Rudin, Leonid I, Stanley Osher, and Emad Fatemi. 1992. Nonlinear total variation based noise removal algorithms. *Physica D: Nonlinear Phenomena* 60(1-4):259–268.

Rudin, Walter, et al. 1964. *Principles of mathematical analysis*, vol. 3. McGraw-hill New York.

Ruiz, Pablo, Xu Zhou, Javier Mateos, Rafael Molina, and Aggelos K Katsaggelos. 2015. Variational bayesian blind image deconvolution: A review. *Digital Signal Processing* 47:116–127.

Saigal, Romesh. 1993. On the inverse of a matrix with several rank one. *Ann Arbor* 1001:48109–2117.

Samangouei, Pouya, Maya Kabkab, and Rama Chellappa. 2018. Defense-gan: Protecting classifiers against adversarial attacks using generative models. *arXiv preprint arXiv:1805.06605*.

Sarlos, T. 2006. Improved approximation algorithms for large matrices via random projections. In *Foundations of computer science (focs)*.

Scharstein, Daniel, Heiko Hirschmüller, York Kitajima, Greg Krathwohl, Nera Nešić, Xi Wang, and Porter Westling. 2014. High-resolution stereo datasets with subpixel-accurate ground truth. In *Pattern recognition*, 31–42. Springer.

Scharstein, Daniel, and Richard Szeliski. 2002. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International journal of computer vision* 47(1-3):7–42.

Schuler, Christian J, Michael Hirsch, Stefan Harmeling, and Bernhard Schölkopf. 2016. Learning to deblur. *IEEE transactions on pattern analysis and machine intelligence* 38(7):1439–1451.

Searcey, Terena, Linda Bierer, and Kenneth L Davis. 1994. A longitudinal study of alzheimerâŁ™s disease: measurement, rate, and predictors of cognitive deterioration. *Am J Psychiatry* 1:51.

Seitz, Steven M, and Simon Baker. 2009. Filter flow. In *Computer vision, 2009 ieee 12th international conference on*, 143–150. IEEE.

Shah, Huma, Kevin Warwick, Jordi Vallverdú, and Defeng Wu. 2016. Can machines talk? comparison of eliza with modern dialogue systems. *Computers in Human Behavior* 58:278–295.

Shamir, Ron. 1987. The efficiency of the simplex method: a survey. *Management Science* 33(3):301–334.

Shapiro, Alexander, Darinka Dentcheva, and Andrzej Ruszczyński. 2009. *Lectures on stochastic programming: modeling and theory*. SIAM.

Shapiro, Stuart C. 1971. A net structure for semantic information storage, deduction and retrieval. Tech. Rep., University of Wisconsin-Madison Department of Computer Sciences.

Shen, Dinggang, and Christos Davatzikos. 2001. Hammer: hierarchical attribute matching mechanism for elastic registration. In *Proceedings ieee workshop on mathematical methods in biomedical image analysis (mmbia 2001)*, 29–36. IEEE.

Soudry, Daniel, and Yair Carmon. 2016. No bad local minima: Data independent training error guarantees for multilayer neural networks. *arXiv:1605.08361*.

Srebro, Nati, Karthik Sridharan, and Ambuj Tewari. 2011. On the universality of online mirror descent. In *Advances in neural information processing systems*, 2645–2653.

Srinivasan, Pratul P., Ren Ng, and Ravi Ramamoorthi. 2017. Light field blind motion deblurring. In *The ieee conference on computer vision and pattern recognition (cvpr)*.

Srivastava, Nitish, et al. 2014. Dropout: a simple way to prevent neural networks from overfitting. *JMLR*.

Stroke, George W, and M Halioua. 1970. A new holographic image deblurring method. *Physics Letters A* 33(1):3–4.

Su, J., D. Vasconcellos Vargas, and S. Kouichi. 2017. One pixel attack for fooling deep neural networks. *ArXiv e-prints*. 1710.08864.

Sun, Jian, Wenfei Cao, Zongben Xu, and Jean Ponce. 2015. Learning a convolutional neural network for non-uniform motion blur removal. In *Proceedings of the ieee conference on computer vision and pattern recognition*, 769–777.

Sun, Libin, Sunghyun Cho, Jue Wang, and James Hays. 2013. Edge-based blur kernel estimation using patch priors. In *Computational photography (iccp), 2013 ieee international conference on*, 1–8. IEEE.

Tai, Cheng, et al. 2015. Convolutional neural networks with low-rank regularization. *arXiv:1511.06067*.

Tao, Xin, Hongyun Gao, Xiaoyong Shen, Jue Wang, and Jiaya Jia. 2018. Scale-recurrent network for deep image deblurring. In *Proceedings of the ieee conference on computer vision and pattern recognition*, 8174–8182.

Taylor, Gavin, et al. 2016. Training neural networks without gradients: A scalable admm approach. In *Icml*.

Tibshirani, Robert. 1996. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)* 267–288.

Tikhonov, Andre Nikolaevich, AV Goncharsky, and M Bloch. 1987. *Ill-posed problems in the natural sciences*.

Triantafillou, Eleni, Tyler Zhu, Vincent Dumoulin, Pascal Lamblin, Kelvin Xu, Ross Goroshin, Carles Gelada, Kevin Swersky, Pierre-Antoine Manzagol, and Hugo Larochelle. 2019. Meta-dataset: A dataset of datasets for learning to learn from few examples. *arXiv preprint arXiv:1903.03096*.

Tsang, I. W., J. T. Kwok, and P-M. Cheung. 2005. Core vector machines: Fast svm training on very large data sets. *Journal of Machine Learning Research*.

Tsybakov, Alexandre B. 2008. *Introduction to nonparametric estimation*. 1st ed. Springer Publishing Company, Incorporated.

Ulvila, Jacob W. 1987. Postal automation (zip+ 4) technology: a decision analysis. *Interfaces* 17(2):1–12.

Vandenberghe, Lieven, V Ragu Balakrishnan, Ragnar Wallin, Anders Hansson, and Tae Roh. Interior-point algorithms for semidefinite programming problems derived from the kyp lemma. In *Positive polynomials in control*, 195–238. Springer.

Vapnik, Vladimir. 1963. Pattern recognition using generalized portrait method. *Automation and remote control* 24:774–780.

Vogel, Curtis R, and Mary E Oman. 1996. Iterative methods for total variation denoising. *SIAM Journal on Scientific Computing* 17(1):227–238.

Wahba, Grace. 1987. Three topics in ill-posed problems. In *Inverse and ill-posed problems*, 37–51. Elsevier.

———. 1990. *Spline models for observational data*, vol. 59. Siam.

Wahba, Grace, Donald R Johnson, Feng Gao, and Jianjian Gong. 1995. Adaptive tuning of numerical weather prediction models: Randomized gcv in three-and four-dimensional data assimilation. *Monthly Weather Review* 123(11):3358–3370.

Wang, Yaqing, and Quanming Yao. 2019. Few-shot learning: A survey. *arXiv preprint arXiv:1904.05046*.

Wang, Yu-Xiang, Veeranjaneyulu Sadhanala, Wei Dai, Willie Neiswanger, Suvrit Sra, and EDU Eric P Xing. 2016. Parallel and distributed block-coordinate frank-wolfe algorithms.

Weizenbaum, Joseph, et al. 1966. Eliza—a computer program for the study of natural language communication between man and machine. *Communications of the ACM* 9(1):36–45.

Wen, Zaiwen, and Wotao Yin. 2013. A feasible method for optimization with orthogonality constraints. *Mathematical Programming* 142(1-2):397–434.

Werner, Alexander, Roberto Lampariello, and Christian Ott. 2012. Optimization-based generation and experimental validation of optimal walking trajectories for biped robots. In *2012 ieee/rsj international conference on intelligent robots and systems*, 4373–4379. IEEE.

White, D. J. 1993. Extension of the Frank-Wolfe algorithm to concave nondifferentiable objective functions. *Journal of Optimization Theory and Applications*.

Wyszecki, Gunter, and Walter Stanley Stiles. 1982. *Color science*, vol. 8. Wiley New York.

Xu, Li, and Jiaya Jia. 2010. Two-phase kernel estimation for robust motion deblurring. In *European conference on computer vision*, 157–170. Springer.

Xu, Li, Jimmy SJ Ren, Ce Liu, and Jiaya Jia. 2014. Deep convolutional neural network for image deconvolution. In *Advances in neural information processing systems*, 1790–1798.

Xue, Hongyang, and Deng Cai. 2016. Stereo matching by joint global and local energy minimization. *arXiv preprint arXiv:1601.03890*.

You, Yu-Li, and Mostafa Kaveh. 1996. A regularization approach to joint blur identification and image restoration. *IEEE Transactions on Image Processing* 5(3): 416–428.

Yu, Rui, Chris Russell, Neill DF Campbell, and Lourdes Agapito. 2015. Direct, dense, and deformable: Template-based non-rigid 3d reconstruction from rgb video. In *2015 ieee international conference on computer vision (iccv)*, 918–926. IEEE.

Yu, Yaoliang, Xinhua Zhang, and Dale Schuurmans. 2017. Generalized conditional gradient for sparse estimation. *The Journal of Machine Learning Research*.

Yuan, G.-X., K.-W. Chang, C.-J. Hsieh, and C.-J. Lin. 2010. A comparison of optimization methods and software for large-scale L1-regularized linear classification. *Journal of Machine Learning Research (JMLR)*.

Zhang, Shengwei, and AG Constantinides. 1992. Lagrange programming neural networks. *IEEE Transactions on Circuits and Systems II*.

Zhou, Zhengyuan, Panayotis Mertikopoulos, Nicholas Bambos, Stephen Boyd, and Peter W Glynn. 2017. Stochastic mirror descent in variationally coherent optimization problems. In *Advances in neural information processing systems*, 7043–7052.

Zhu, Xiang, and Peyman Milanfar. 2011. Restoration for weakly blurred and strongly noisy images. In *Applications of computer vision (wacv), 2011 ieee workshop on*, 103–109. IEEE.