NORTHWESTERN UNIVERSITY

Second-Order Methods for Stochastic and Nonsmooth Optimization

A DISSERTATION

SUBMITTED TO THE GRADUATE SCHOOL

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

for the degree

DOCTOR OF PHILOSOPHY

Field of Industrial Engineering and Management Sciences

By

Nitish Shirish Keskar

EVANSTON, ILLINOIS

June 2017

ABSTRACT

Second-Order Methods for Stochastic and Nonsmooth Optimization

Nitish Shirish Keskar

The goal of this thesis is to design practical algorithms for nonlinear optimization in the case when the objective function is stochastic or nonsmooth. The thesis is divided into three chapters. Chapter 1 describes an active-set method for the minimization of an objective function that is structurally nonsmooth, viz., it is the sum of a smooth convex function and an ℓ_1-regularization term. Problems of this nature primarily arise, e.g., in machine learning, when sparse solutions are desired. A distinctive feature of the method is the way in which active-set identification and second-order subspace minimization steps are integrated to combine the predictive power of the two approaches. At every iteration, the algorithm selects a candidate set of free and fixed variables, performs an (inexact) subspace phase, and then assesses the quality of the new active set. If it is not judged to be acceptable, then the set of free variables is restricted and a new active-set prediction is made. We establish global convergence for our approach, and compare an implementation of the new method against state-of-the-art numerical codes to demonstrate its competitiveness.

Chapter 2 outlines an algorithm for minimizing a continuous function that may be nonsmooth and nonconvex, subject to bound constraints. We propose an algorithm that uses the L-BFGS quasi-Newton approximation of the problem's curvature together with a variant of a weak Wolfe line search. The key ingredient of the method is an active-set selection strategy that defines the subspace in which search directions are computed. To overcome the inherent shortsightedness of the gradient for a nonsmooth function, we propose two strategies. The first relies on an approximation of the ϵ-minimum norm subgradient, and the second uses an iterative corrective loop that augments the active set based on the resulting search directions. We describe a Python implementation of the proposed algorithm and present numerical results on a set of standard test problems to illustrate the efficacy of our approach.

Chapter 3 investigates the gap in statistical generalization performance between large- and small-batch methods in the task of training state-of-the-art deep neural network models. The stochastic gradient descent (SGD) method and its variants are algorithms of choice for many Deep Learning tasks. These methods operate in a small-batch regime wherein a fraction of the training data, say 32–512 data points, is sampled to compute an approximation to the gradient. It has been observed in practice that when using a larger batch there is a degradation in the quality of the model, as measured by its ability to generalize. We investigate the cause for this generalization drop in the large-batch regime and present numerical evidence that supports the view that large-batch methods tend to converge to sharp minimizers of the training and testing functions — and as is well known, sharp minima lead to poorer generalization. In contrast, small-batch methods consistently converge to flat minimizers, and our experiments support a commonly held view that this

is due to the inherent noise in the gradient estimation. We also discuss several strategies to attempt to help large-batch methods eliminate this generalization gap.

Acknowledgements

Before anyone else, I have to express my sincerest gratitude to Prof. Andreas Waechter and Prof. Jorge Nocedal for their guidance throughout my PhD. I consider myself lucky to have not just one but two excellent advisors who, in their almost orthogonal approaches, helped me grow as a researcher. With Andreas focus on specificity and precision, and Jorges emphasis on broad, albeit unstructured, ideas, I truly got the best of both worlds. Eventually, I might forget the update rule for BFGS (unlikely) but I dont foresee forgetting the essential skills of research, communication, and learning you taught me. I would also like to thank Prof. Frank Curtis and Prof. Ermin Wei for serving as my thesis and prospectus committee members. I learnt a great deal from our interactions! Ive had the good fortune of learning from, and assisting, some excellent teachers at Northwestern for which I am greatly thankful for. Over the course of my PhD, Ive also had the privilege of collaborating with many inspiring researchers from both academia and industry including Figen Oztoprak, Richard Byrd, George Saon, Mikhail Smelyanskiy, Dheevatsa Mudigere and Peter Tang. My time at Northwestern wouldnt be same without my group-mates: Gillian, Travis, Sammy, Stefan, Alvaro, Ben, Francisco, Albert, Vijaya, Samira and Alejandra. I dont think I could have asked for a more hard-working, good-natured, supportive, and light-hearted group to spend five years with. Im never going to forget times spent in the barbeques, trivia quizzes, side projects, gossiping, group meetings and practical jokes. The years I spent in L375 were some of the best, most productive, and

laughter-packed years of my life; Im going to miss them and I hope all the rituals (including pranks) continue. Id also like to thank my IEMS cohort for getting me through the first year and the entire IEMS staff for all their help with my, at times incessant, inquiries. Id be remiss if I didnt give a shoutout to the Bay Area Doges for their help and friendship whenever I ran off to San Francisco for a break.

Mom, Dad, Manjiri, Nikhil, Anya, my family-in-law and Sharvari, I couldnt have done this without your support and love, and for that I will always be grateful. Sharvari, Im especially grateful to you for listening to me complain and rhapsodize through the innumerable ups-and-downs of my PhD journey.

Table of Contents

List of Tables

List of Figures

CHAPTER 1

Introduction

Many problems from fields such as machine learning, statistics, and control can be solved using techniques from Nonlinear Optimization. Generally, these problems can be written as

$$\min_{w \in C} \quad f(w) \tag{1.1}$$

where f is the objective being minimized, w is the design variable and C is a constraint set that could be simple (such as points satisfying bounds or non-negativity) or more complex. In this thesis, we consider the problem of solving (1.1) in contexts when f is either nonsmooth or stochastic. Specifically, we consider the optimization problems arising in large-scale stochastic non-convex optimization, e.g., in deep learning, and of unconstrained structured and bound-constrained unstructured nonsmooth optimization. In order to achieve optimum performance, the underlying structure of the function must be exploited by the optimization algorithm. We briefly motivate our investigation, and conclude this chapter by discussing the organization of this thesis.

Stochasticity

A generic formulation of unconstrained stochastic optimization problems can be written as (1.1) with

$$f(w) = \mathbb{E}_{\xi}[G(w, \xi)], \tag{1.2}$$

where $\mathbb{E}$ denotes an expectation and G is any function with dependence on the design variable w and a random vector ξ (Homem-de Mello and Bayraksan, 2014). In this thesis, we consider stochastic optimization problems that arise in the context of supervised machine learning. The goal is to learn a model ϕ_w that, given a data point $x \in \mathbb{R}^n$, predicts a label $y \in \mathbb{R}$. The subscript w indicates dependence of the model ϕ to a set of parameters $w \in \mathbb{R}^d$. This dependence could lead to a convex ϕ_w as in the case of logistic regression or least-squares regression, or non-convex as in the case of deep learning. Supervised learning problems of this kind form an important pillar of machine learning and arise in Speech Recognition, Computer Vision, and Natural Language Processing (Bengio et al., 2016). In order to learn the model ϕ_w, the first step often involves the design of a loss function l that captures the difference between the true label and the label predicted by the model ϕ_w. Popular choices for the loss include quadratic functions for regression and the cross-entropy function for classification. The process of training (or learning) the model ϕ_w is equivalent to solving the following optimization problem.

$$\min_{w} \quad \mathbb{E}_{(x,y)\sim\mathcal{D}}\left[l(\phi_w(x), y)\right]. \tag{1.3}$$

Note that the data (x, y) is considered to be random and takes on the role of ξ as in the generic formulation (1.2). However, the distribution $\mathcal{D}$ from which the data (x, y) is generated is rarely known. Instead, all that is available is a set of m data points (x_i, y_i) for $i = \{1, 2, \cdots, m\}$. We can then replace problem (1.3) by its finite-sample (also called the sample average) approximation

$$\min_{w} \quad \frac{1}{m} \sum_{i=1}^{m} f_i(w), \tag{1.4}$$

where $f_i(w) = [l(\phi_w(x_i), y_i)]$.

In most state-of-the-art machine learning models that we consider, all of $\{m, n, d\}$ are large. In this case, the computational cost of popular classical optimization algorithms such as gradient descent, BFGS or Newton's method (Nocedal and Wright, 2006) are prohibitive, and stochastic algorithms such as stochastic gradient descent (SGD), are often effective (Bottou et al., 2016). Denoting the k^{th} iterate as w_k, the step size at that iteration as α_k, and the gradient of f at w_k by $\nabla f(w_k)$, gradient descent would update iterates using the rule

$$w_{k+1} = w_k - \frac{\alpha_k}{m} \sum_{i=1}^{m} \nabla f_i(w_k)$$

In contrast, (mini-batch) SGD would take steps of the form

$$w_{k+1} = w_k - \frac{\alpha_k}{|B_k|} \sum_{i \in B_k} \nabla f_i(w_k) \tag{1.5}$$

where the batch $B_k \subseteq \{1, 2, \cdots, m\}$ and typically contains only 32–512 data points which are chosen at random for each k. SGD, and its variants like Adam (Kingma and Ba, 2015)

and Adagrad (Duchi et al., 2011), have been used for achieving state-of-the-art results on several machine learning problems (Bengio et al., 2016). However, many questions regarding the scalability and convergence of these algorithms remain to be adequately answered.

Nonsmoothness

Consider the general problem (1.1) without any constraints. When f is twice continuously differentiable (i.e., $\mathcal{C}^2$-smoothness), first- and second-order methods can be employed for optimizing f. First-order methods include gradient descent and its variants while second-order methods include algorithms of the quasi-Newton and Newton families. Under assumptions of $\mathcal{C}^2$-smoothness and the presence of a globalization mechanism such as a line search, such algorithms are globally convergent and achieve a rapid local convergence rate (Nocedal and Wright, 2006). However, if the function is non-differentiable, these algorithms are typically not applicable, and even if they are, they will often fail. The problem could be further exacerbated if (1.1) also contains constraints. Such non-differentiable optimization problems are common in various fields including statistics, machine learning, economics and mechanical engineering (Bagirov et al., 2014). The nonsmoothness is either structured, where the manifold on which the function is nonsmooth is explicitly known a-priori, or unstructured, where no such identifying characteristics are known a-priori. When the nonsmoothness is structured, it may be possible to exploit this structure when designing algorithms. For instance, problems of the form $\min_x(f(x) + g(x))$ with a smooth and convex function f, and nonsmooth and convex function g can be solved effectively if the proximal operator for g is easily computable (Boyd and Vandenberghe, 2004). Problems

of this kind arise in statistics and machine learning where sparse (or more generally, structured) solutions are desired (Friedman et al., 2001). However, the structure of the nonsmoothness may not always be known and thus, general-purpose algorithms are needed.

Overview

In this thesis, we investigate second-order algorithms for solving problems when the objective function f may be nonsmooth or stochastic. The thesis is divided into three chapters. In the first chapter, we discuss an algorithm for solving a problem with structured nonsmoothness, viz., an objective function composed of a smooth convex function and an ℓ_1-norm penalty term. These problems are common when sparse solutions are desired for machine learning models such as logistic regression or least-squares regression (Friedman et al., 2001). A distinctive feature of the proposed algorithm is the interweaving of the step computation and active-set prediction mechanisms. At every iteration, the algorithm selects a candidate active set and revises the selection recursively until it is deemed acceptable. We prove that the algorithm possesses global convergence guarantees and numerically demonstrate its efficacy. The second chapter deals with the problem of unstructured nonsmoothness. We propose an algorithm for solving (1.1) in the presence of bound constraints, under mild assumptions on f. Specifically, we assume that f is continuous on its domain and is differentiable at every trial point, but place no assumptions on the convexity of f. We propose an algorithm that uses the L-BFGS quasi-Newton approximation of the problem's Hessian together with a variant of a weak Wolfe line search. To overcome the inherent shortsightedness of negative gradient steps for nonsmooth functions, we propose two strategies. The first relies on an approximation of the ϵ-minimum

norm subgradient, and the second uses an iterative corrective loop that augments the active set based on the resulting search directions. We describe our Python implementation and demonstrate the efficacy of our approach through numerical experiments on standard test problems. Finally, in Chapter 3, we investigate the gap in generalization performance between large- and small-batch methods in the task of training state-of-the-art deep neural network models. Larger batch sizes provide more parallelism opportunities since the gradient computation can be effectively distributed. However, it has been observed in practice that when using a larger batch, there is a degradation in the quality of the model, as measured by its ability to generalize. In this context, generalization implies the performance of the trained model on unseen data. We investigate the cause for this generalization drop and present numerical evidence that supports the view that large-batch methods tend to converge to sharp minimizers of the training and testing functions — and as is empirically observed, sharp minima lead to poorer generalization. In contrast, small-batch methods consistently converge to flat minimizers, and our experiments support a commonly held view that this is due to the inherent noise in the gradient estimation. We also discuss several strategies to attempt to help large-batch methods eliminate this generalization gap.

In the chapters to follow, we describe three projects in detail. Each chapter represents a separate publication and is self-contained. We introduce the required notation at the beginning of each chapter and also include hyperlinks to the code needed to reproduce our numerical experiments.

CHAPTER 2

A Second-Order Method for Convex ℓ_1-Regularized Optimization with Active-Set Prediction

2.1. Introduction

The problem of minimizing an objective that is the sum of a smooth convex function and a regularization term has received much attention; see e.g. (Sra et al., 2011; Bach et al., 2012) and the references therein. This problem arises in statistics, machine learning and in many other areas of applications. In this chapter we focus on the case when the regularizer is defined in terms of an ℓ_1-norm, and propose an algorithm that employs a recursive active-set selection mechanism designed to make a good prediction of the active subspace at each iteration. Here, the term active-set refers to the variables that would be held at 0. This mechanism combines first- and second-order information, and is designed with the large-scale setting in mind.

The problem under consideration is given by

$$\min_{x \in \mathbb{R}^n} \ \phi(x) = f(x) + \mu \|x\|_1. \tag{2.1}$$

We assume that f is a smooth convex function and $\mu > 0$ is a fixed penalty parameter.

The algorithm proposed in this chapter is different in nature from the most popular methods proposed for solving problem (2.1). These include first-order methods, such as ISTA, SpaRSA and FISTA (Donoho, 1995; Wright et al., 2009; Beck and Teboulle, 2009),

and proximal Newton methods that compute a step by minimizing a piecewise quadratic model of (2.1) using (for example) a coordinate descent iteration (Yuan et al., 2012; Tseng and Yun, 2009; Hsieh et al., 2011; Scheinberg and Tang, 2014; Byrd et al., 2015; J. et al., 2014; Schmidt et al., 2011; Olsen et al., 2012). The proposed algorithm also differs from methods that solve (2.1) by reformulating it as a bound constrained problem (Wen et al., 2010; Fountoulakis et al., 2014; Koh et al., 2007; Schmidt et al., 2007; Schmidt, 2010), and from recent methods that are specifically designed for the case when f is a convex quadratic (Solntsev et al., 2014; Wen et al., 2010; De Santis et al., 2014).

Our algorithm belongs to the class of *orthant-based methods* (Andrew and Gao, 2007; Byrd et al., 2012b) that minimize a smooth quadratic model of ϕ on a sequence of orthant faces of $\mathbb{R}^n$ until the optimal solution is found. Every iteration of the algorithm consists of a *corrective cycle* of orthant-face predictions and subspace minimization steps. This cycle is terminated when the orthant-face prediction is deemed to be reliable. (A variant of this idea has been employed in (Byrd et al., 2012a) and (Hungerländer and Rendl, 2015).) After a trial iterate has been computed, a globalization mechanism accepts or modifies it (if necessary) to ensure overall convergence of the iteration.

The idea of employing a correction mechanism for refining the selection of the orthant face was introduced in (Byrd et al., 2012a) for the case when f is a convex quadratic function. That algorithm is, however, not competitive with state-of-the-art methods in terms of CPU time because each iteration requires the exact solution of a subspace problem, which is expensive, and because the orthant-face prediction mechanism is too liberal (an observation also made in (Han and Curtis, 2015)) and can lead to long corrective cycles. These deficiencies are overcome in our algorithm, which introduces two key components.

We employ an adaptive filtering mechanism that in conjunction with the corrective cycle yields an efficient prediction of zero variables at each iteration. We also design a strategy for solving, inexactly, the subproblems arising during each corrective step in a way that does not degrade the accuracy of the orthant-face prediction and yields important savings in computation. We show that the algorithm is globally convergent for strongly convex problems. Numerical tests on a variety of machine learning data sets suggest that our algorithm is competitive with a leading state-of-the-art code.

The main features of our algorithm can also be highlighted by contrasting them with recently proposed proximal Newton methods for solving problem (2.1). The algorithms proposed by (Yuan et al., 2012; Scheinberg and Tang, 2014; Hsieh et al., 2011) and others first chose an active set of variables using first-order sensitivity information. The active variables are set to zero, and the rest of the variables are updated by minimizing a *piecewise quadratic* approximation to (2.1) given by

$$q^k(x) = f(x^k) + (x - x^k)^T \nabla f(x^k) + \frac{1}{2}(x - x^k)^T \nabla^2 f(x^k)(x - x^k) + \mu \|x\|_1. \tag{2.2}$$

This minimization is performed inexactly using a randomized coordinate descent method. After a trial iterate is computed in this manner, a backtracking line search is performed to ensure decrease in $\phi(x)$.

The proximal Newton methods just outlined employ a very simple mechanism to determine the set of active variables at each iteration, namely the minimum norm subgradient. On the other hand, they solve the sophisticated lasso subproblem (2.2) that inherits the non-smooth structure of the original problem and permits iterates to cross points of non-differentiability of $\phi(x)$. The latter property allows proximal Newton methods to

refine the active set with respect to its initial choice. In contrast, our method invests a significant amount of computation in the identification of a working orthant face in $\mathbb{R}^n$, and then minimizes the following simple smooth quadratic approximation of the problem on that orthant face,

$$\bar{q}^k(x) = f(x^k) + (x - x^k)^T \nabla f(x^k) + \frac{1}{2}(x - x^k)^T \nabla^2 f(x^k)(x - x^k) + \mu \zeta^T x, \tag{2.3}$$

where ζ is an indicator with values 0, 1 or -1, that identifies the orthant face. The working orthant is selected carefully, by verifying that the predictions made at each corrective step are realized. We do so because a simpler selection of the orthant face, such as that performed in the OWL method (Andrew and Gao, 2007), or the method described in (Byrd et al., 2012b) can generate poor steps in some circumstances.

Given that the two approaches (proximal Newton with coordinate descent solver and our proposed method) are based on different principles, it is natural to ask if one of them will emerge as the preferred second-order technique for the solution of problem (2.1). To answer this question we compare a MATLAB implementation of our approach on binary classification problems with the well-known solver LIBLINEAR (written in C), based on CPU time. One of the main conclusions of this chapter is that *both approaches* have their strengths. We also report comparisons with the primal-dual Newton conjugate gradient method (pdNCG) (Fountoulakis and Gondzio, 2015), which shows its strength on large and ill-conditioned data sets.

Orthant-based methods have the attractive property that the subspace minimization can be performed by a direct linear solver or by an iterative method such as the conjugate gradient method, which is efficient on a wide range of applications. On the other hand,

the proximal Newton approach (with coordinate descent solver) is very effective on applications where the Hessian matrix is diagonally dominant (or nearly so). We note that the observations made in this chapter about proximal Newton methods pertain only to those employing coordinate descent as the subproblem solver. This is at present the subproblem solver of choice and has been implemented in software packages such as LIBLINEAR (Yuan et al., 2012), LHAC (Scheinberg and Tang, 2014) and QUIC (Hsieh et al., 2011). However, other subproblem solvers could be used, and in that case the numerical performance of the proximal Newton method would depend on the characteristics of that solver.

Orthant-based methods and proximal Newton methods both share the need for effective criteria for deciding when an approximate solution of the subproblem is acceptable. Implementations of the proximal Newton method employ adaptive techniques (heuristics or rules based in randomized analysis) (Yuan et al., 2012; Scheinberg and Tang, 2014; Hsieh et al., 2011), or rules based on an optimality measure (Byrd et al., 2015; J. et al., 2014). Our implementation makes use of the classic termination criteria based on the relative error in the residue of the linear system (Nocedal and Wright, 2006).

This chapter is organized in 5 sections. In Section 2.2 we outline the algorithm, paying particular attention to the orthant-face identification mechanism. Section 2.3 discusses the procedure by which we safeguard against poor steps and ensure global convergence of the algorithm. In Section 2.4, we present a comparison of our algorithm against the state-of-the-art code LIBLINEAR for the solution of binary classification problems; some final remarks are made in Section 2.5.

2.2. The Proposed Algorithm

The algorithm exploits the fact that the objective function ϕ is smooth in any *orthant face* of $\mathbb{R}^n$, which is defined as the intersection of an orthant in $\mathbb{R}^n$ and a subspace $\{x : x_i = 0,\, i \in I \subset \{1, \ldots n\}\}$.

At every iteration, the algorithm identifies an orthant face in $\mathbb{R}^n$ using sensitivity information, performs a minimization on that orthant face to produce a trial point, refines the orthant-face selection (if necessary), and repeats the process until the choice of the orthant face is judged to be acceptable. Upon termination of this cycle, a backtracking line search is performed where the trial points are projected onto the active orthant.

To describe the algorithm in detail, we introduce some notation. Let $g(x)$ denote the minimum norm subgradient of the objective function (2.1) at a point x. Thus, we have

$$g_i(x) = \begin{cases} \nabla_i f(x) + \mu \text{ if } x_i > 0 \text{ or } (x_i = 0 \text{ and } \nabla_i f(x) + \mu < 0) \\ \nabla_i f(x) - \mu \text{ if } x_i < 0 \text{ or } (x_i = 0 \text{ and } \nabla_i f(x) - \mu > 0) \\ 0 \qquad\qquad \text{otherwise,} \end{cases} \tag{2.4}$$

for $i = 1, \ldots n$, where

$$\nabla_i f(x) \stackrel{\text{def}}{=} \frac{\partial f(x)}{\partial x_i}.$$

At an iterate x^k, we define three sets:

$$\mathcal{A}^k = \{i |\, x_i^k = 0 \text{ and } |\nabla_i f(x^k)| \leq \mu\} \tag{2.5}$$

$$\mathcal{F}^k = \{i |\, x_i^k \neq 0\} \tag{2.6}$$

$$\mathcal{U}^k = \{i |\, x_i^k = 0 \text{ and } |\nabla_i f(x^k)| > \mu\}. \tag{2.7}$$

The variables in $\mathcal{A}^k$ are kept at zero (since the corresponding components of $g_i(x^k)$ are zero), while those in $\mathcal{F}^k$ are free to move. The remaining variables are in the set $\mathcal{U}^k$. The decision of which of these are allowed to move significantly impacts the efficiency of the algorithm. Using the *selection mechanism* described below, we first create a partition of $\mathcal{U}^k$,

$$\mathcal{U}^k = \mathcal{U}_A \cup \mathcal{U}_F, \tag{2.8}$$

where the variables in $\mathcal{U}_A$ are fixed at zero and the variables in $\mathcal{U}_F$ are allowed to move. We then update the active set as

$$\mathcal{A}^k \leftarrow \mathcal{A}^k \cup \mathcal{U}_A, \tag{2.9}$$

and compute a trial step d^k as the (approximate) solution of the smooth quadratic problem

$$\begin{aligned} \min_{d \in \mathbb{R}^n} \quad & \psi(d) = d^T g(x^k) + \frac{1}{2} d^T H^k d \\ \text{s.t.} \quad & d_i = 0, \ i \in \mathcal{A}^k, \end{aligned} \tag{2.10}$$

where $H^k = \nabla^2 f(x^k)$. The trial iterate is defined as

$$\hat{x}^k = x^k + d^k.$$

We then start the *corrective cycle* and check whether all variables in the set $\mathcal{U}_F$ moved as predicted; i.e., whether

$$\operatorname{sgn}([\hat{x}^k]_i) = \operatorname{sgn}(-[g(x^k)]_i) \quad \text{for all } i \in \mathcal{U}_F, \tag{2.11}$$

where $\text{sgn}(0) \stackrel{\text{def}}{=} 0$. Any variable $j \in \mathcal{U}_F$ for which this equality does not hold, is removed from the set $\mathcal{U}_F$ and added to $\mathcal{U}_A$. The set $\mathcal{A}^k$ is then updated according to (2.9) and a new trial step is recomputed by solving (2.10). We repeat this corrective cycle until all predictions are correct and the trial point $\hat{x}^k$ satisfies (2.11).

The algorithm then performs a projected backtracking line search along d^k to ensure that the resulting point yields a decrease in the piecewise quadratic model $q^k(x)$ defined in (2.2). (We do not perform the line search on the objective function (2.1) as that is often more expensive, the repercussions of this choice are discussed at the end of this section.)

At iteration k, we identify the current orthant face based on sensitivity information (2.4) and define the vector ζ^k by

$$\zeta_i^k = \begin{cases} \text{sgn}([x^k]_i) & \text{if } [x^k]_i \neq 0 \\ \text{sgn}(-[g(x^k)]_i) & \text{if } [x^k]_i = 0. \end{cases} \tag{2.12}$$

Let $\mathcal{P}^k(x)$ be the projection operator that projects $x \in \mathbb{R}^n$ onto the orthant defined by ζ^k; i.e.,

$$\mathcal{P}_i^k(x) = \begin{cases} x_i & \text{if } \text{sgn}(x_i) = \zeta_i^k \\ 0 & \text{otherwise.} \end{cases} \tag{2.13}$$

We then search for the largest step size $\alpha \in \{2^0, 2^{-1}, 2^{-2}, \cdots\}$ such that

$$q(x^k) \geq q(\mathcal{P}^k(x^k + \alpha \cdot d^k)),$$

where q is the non-smooth quadratic approximation given by (2.2). Such a step size exists because d^k is a descent direction for the smooth quadratic function $\bar{q}^k$ and because the trial point lies within the orthant defined by ζ^k for sufficiently small steps (see Theorem 2.6.4).

Before giving a detailed description of the algorithm, we describe the *selection mechanism* that, at the beginning of each corrective cycle, defines the splitting (2.8) of the set $\mathcal{U}^k$ into variables $\mathcal{U}_A$, that are kept at zero, and variables $\mathcal{U}_F$, that are allowed to move.

At the start of the algorithm, we select a scalar $\eta \in (0, 1)$ and set $|\mathcal{U}_F| = \tau^0 \stackrel{\text{def}}{=} \lfloor \eta \times n \rfloor$; i.e., the cardinality of the set $\mathcal{U}_F$ is a fraction of the dimension of the problem. On subsequent iterations, we update the parameter τ^k based on its previous value τ^{k-1} and the number of iterations in the previous corrective cycle. If there were no corrections in the previous corrective cycle, we set $\tau^{k+1} = 2\tau^k$ to allow more variables to change at the next outer iteration; otherwise we keep the value of τ^k unchanged. Since the number of variables in $\mathcal{U}_F$ cannot be larger than $|\mathcal{U}^k|$, the actual size of $\mathcal{U}_F$ is given by

$$|\mathcal{U}_F| = \hat{\tau}^k \stackrel{\text{def}}{=} \min(|\mathcal{U}^k|, \tau^k).$$

We use a greedy strategy to populate the sets $\mathcal{U}_A$ and $\mathcal{U}_F$: we collect in $\mathcal{U}_F$ the $\hat{\tau}^k$ variables in $\mathcal{U}^k$ with the largest components of the subgradient $|g(x^k)|$. Thus, for any $i \in \mathcal{U}_F$ and $j \in \mathcal{U}_A$, we have $|g_i(x^k)| \geq |g_j(x^k)|$.

A formal description of the overall method is given in Algorithm 1.

Algorithm 1 Preliminary Adaptive Orthant-Based Method

Given $x^0 \in \mathbb{R}^n$, $\mu > 0$, $\eta \in (0, 1)$.

Let $\tau^0 = \lfloor \eta \times n \rfloor$.

while $k = 0, 1, 2, \cdots$ and stopping criterion not met **do**

Active-Set Identification:

$$\mathcal{A}^k = \{i | (x_i)^k = 0 \text{ and } |\nabla_i f(x^k)| \leq \mu\}$$

$$\mathcal{F}^k = \{i | (x_i)^k \neq 0\}$$

$$\mathcal{U}^k = \{i | (x_i)^k = 0 \text{ and } |\nabla_i f(x^k)| > \mu\}$$

Selection Mechanism:

Compute $g(x^k)$ by (2.4) and ζ^k by (2.12).

Set $\hat{\tau}^k \leftarrow \min(|\mathcal{U}^k|, \tau^k)$.

Choose $\mathcal{U}_F, \mathcal{U}_A \subseteq \mathcal{U}^k$ such that $\mathcal{U}_F \cap \mathcal{U}_A = \emptyset$, $|\mathcal{U}_F| = \hat{\tau}^k$ and for any $i \in \mathcal{U}_F$ and $j \in \mathcal{U}_A$, $|g_i(x^k)| \geq |g_j(x^k)|$.

Set $\mathcal{A}^k \leftarrow \mathcal{A}^k \cup \mathcal{U}_A$.

Compute or update second-order approximation H^k.

Corrective Cycle:

Set $V^k \leftarrow \mathcal{U}_F$ and $j \leftarrow 0$.

while $V^k \neq \emptyset$ **do**

$$d^k = \underset{d_i = 0, i \in \mathcal{A}^k}{\arg\min} \, d^T g(x^k) + \tfrac{1}{2} d^T H^k d$$

Set $\hat{x}^k \leftarrow x^k + d^k$.

Set $V^k = \{i \in \mathcal{U}_F \setminus \mathcal{A}^k | \zeta_i^k \neq \text{sgn}(\hat{x}_i^k)\}$.

Set $\mathcal{A}^k \leftarrow \mathcal{A}^k \cup V^k$ and $j \leftarrow j + 1$.

end while

Algorithm 1 Preliminary Adaptive Orthant-Based Method

if $j = 1$ **then**

  Set $\tau^{k+1} = 2 \cdot \tau^k$.

end if

Projected Line Search:

Set $\alpha \leftarrow 1$.

while $q(x^k) > q(\mathcal{P}^k(x^k + \alpha \cdot d^k))$ **do**

  Set $\alpha \leftarrow \alpha/2$.

end while

Set $x^{k+1} = \mathcal{P}^k(x^k + \alpha \cdot d^k)$.

end while

In this chapter we assume that the quadratic model (2.10) employs exact Hessian information, i.e. $H^k = \nabla^2 f(x^k)$, and that we perform an approximate minimization of this problem using the conjugate gradient method in the appropriate subspace of dimension $(n - |\mathcal{A}^k|)$; see e.g., (Nocedal and Wright, 2006). The matrix H^k can also be defined by quasi-Newton updates, specifically using the compact representations of limited-memory BFGS matrices (Byrd et al., 1994a). Although we do not explore a quasi-Newton variant in this chapter, we expect it to be effective in many applications. Further, we use a stopping criterion identical to the one used in (Byrd et al., 2015); i.e., the algorithm terminates when the semi-smooth optimality measure as defined in that chapter falls below a set tolerance.

Our *selection mechanism* for defining the splitting (2.8) is motivated by the following considerations. If all variables in $\mathcal{U}^k$ were allowed to move, the algorithm would have similar properties to the OWL method (Andrew and Gao, 2007), whose performance is not uniformly successful (see Section 2.4). Indeed, we observed a more reliable performance

when the size of $\mathcal{U}_F$ is limited. This also has computational benefits because the subproblem (2.10) is typically less expensive to solve when the number of free variables is smaller (i.e., when the set $\mathcal{A}^k$ is larger). On the other hand, in the extreme case $|\mathcal{U}_F| = 1$, the algorithm resembles a classical active-set method, which is not well-suited for large-scale problems.

These trade-offs are addressed by the dynamic strategy employed in steps 5 and 17 of Algorithm 1. Initially, we choose $\mathcal{U}_F$ to be a small subset of $\mathcal{U}^k$ (by selecting η to be small). The algorithm increases the size of $\mathcal{U}_F$ in subsequent iterations if there is evidence that the current choice is too restrictive. As indicator we observe the number of iterations in the previous corrective cycle. A small number of corrections (in our implementation this number is 1) suggests that the choice of $\mathcal{U}_F$ may be too conservative and the size of $\mathcal{U}_F$ is doubled at the next outer iteration. We have found that this selection mechanism leads to more gradual and controlled changes in the active set compared to other orthant-based methods like OWL and the method proposed in Section 5 of (Byrd et al., 2012b). The strategy for gradually growing the set of free variables is also employed in (Han and Curtis, 2015) in the context of isotonic regression and trend filtering.

The projected backtracking line search in steps 19–22 of Algorithm 1 serves two main purposes: it enforces sparsity in the iterates and ensures quality of the steps (by reducing the model objective function value). Because the overall convergence of our method is driven by a model-based mechanism described in the next section, it is not necessary to evaluate the original objective function at all trial points. Instead, the line search is based on the quadratic model, as this saves potentially expensive function evaluations. In practice, the progress predicted by the quadratic model is so reliable that a further

refinement of the step by the globalization procedure described in the next section is rarely needed and was in fact, never required in our experiments.

2.3. Globalization Strategy

While Algorithm 1 generally works well in practice, it may fail (cycle) when the changes in the active set are not sufficiently controlled. By adding a globalization mechanism we ensure that all iterates generated by the algorithm provide sufficient reduction in the objective function and converge to the solution. Our mechanism employs the iterative soft-thresholding algorithm (ISTA) (Donoho, 1995; Daubechies et al., 2004) to generate a reference point. Because the ISTA method enjoys a global linear rate of convergence on strongly convex problems, it provides a benchmark for the progress of our algorithm.

We modify Algorithm 1 as follows. The iterate computed in line 23 is now regarded as a trial iterate and denoted by $\hat{x}^k$. To decide if this point is acceptable we check whether it produces a lower function value than the ISTA step computed from the starting point of the iteration, x^k. If so, we accept the trial point; otherwise, we search along the segment joining $\hat{x}^k$ and the ISTA point x^k_{ISTA} to find an acceptable point. Given a Lipschitz constant L for the gradient of f, the cost of computing the ISTA step is negligible since gradient information is already available at x^k. However, the evaluation of $\phi(x^k_{\text{ISTA}})$ incurs an additional cost. To get around this expense, we use the value of an upper quadratic approximation of ϕ at x^k_{ISTA} as a surrogate to $\phi(x^k_{\text{ISTA}})$. More specifically, assuming that L is a Lipschitz constant of ∇f, we define the value of the surrogate function as

$$\Gamma^k = f(x^k) + \nabla f(x^k)^T(x^k_{\text{ISTA}} - x^k) + \tfrac{L}{2}\|x^k_{\text{ISTA}} - x^k\|_2^2 + \mu\|x^k_{\text{ISTA}}\|_1. \tag{2.14}$$

The computation of Γ^k requires only one inner product. The complete version of the algorithm, including the globalization mechanism, is given in Algorithm 2.

Algorithm 2 Orthant-Based Adaptive Method (OBA)

Given $x^0 \in \Re^n$, $L > 0$, $\mu > 0$, $\eta \in (0, 1)$ and $\epsilon > 0$.
Let $\tau^0 = \lfloor \eta \times n \rfloor$
while $k = 0, 1, 2, \cdots$ and stopping criterion not met **do**
    Carry out steps 1 – 22 of Algorithm 1.
    Set $\hat{x}^k = \mathcal{P}^k(x^k + \alpha \cdot d^k)$.
    Globalization:
    Compute ISTA step at x^k as

$$x^k_{\text{ISTA}} = \mathcal{S}_{\mu/L}(x^k - \tfrac{1}{L}\nabla f(x^k))$$

    where $\mathcal{S}_\alpha(x)$ is a component-wise operator defined as $\mathcal{S}_\alpha(x)_i = \max\{|x_i| - \alpha, 0\} \cdot \text{sgn}(x_i)$.
    Set $\bar{d}^k \leftarrow \hat{x}^k - x^k_{\text{ISTA}}$ and $\bar{\alpha} \leftarrow 1$.
    Calculate Γ^k using (2.14).
    while $\phi(x^k_{\text{ISTA}} + \bar{\alpha} \cdot \bar{d}^k) > \Gamma^k$ **do**
        Set $\bar{\alpha} \leftarrow \bar{\alpha}/2$.
        if $\bar{\alpha} < \epsilon$ **then**
            Set $\bar{\alpha} \leftarrow 0$.
        end if
    end while
    Set $x^{k+1} = x^k_{\text{ISTA}} + \bar{\alpha} \cdot \bar{d}^k$.
end while

The following convergence result is proven in Section 2.6.

Theorem. *Assume that f is continuously differentiable and strongly convex and that ∇f is Lipschitz continuous. Then, the iterates $\{x^k\}$ generated by Algorithm 2 converge to the optimal solution $x^\star$ of problem* (2.1) *at a linear rate.*

2.4. Numerical Experiments

In this section, we demonstrate the viability of our approach. While our method applies to any convex function with an additive ℓ_1-regularizer, we focus on the specific problem of binary classification using logistic regression. This problem is well studied (Friedman et al., 2010) with many data sets of varying sizes, structures and fields of study. We should note that the results reported on this problem are representative of the performance of OBA on other functions f(including multi-class logistic regression, probit regression and LASSO) where similar trends are observed.

The data sets chosen for the numerical tests are listed in Table 2.1. Synthetic is a randomly generated, non-diagonally dominant problem described in Section 2.7. Alpha is a data set from the Pascal Large Scale Learning Challenge (Pascal Large Scale Learning Challenge, 2008). We also include two ill-conditioned variants, IC-Alpha and IC-Random, which have (Hessian) condition numbers of the order of 10^{10}. IC-Alpha is generated by artificially increasing the condition number of the Alpha data set while IC-Random is generated using the procedure OsGen (Fountoulakis and Gondzio, 2015). The features of these data sets have been normalized to lie in the range $[-1, 1]$. Details for the other data sets along with their preprocessing steps can be found in `http://www.csie.ntu.edu.tw/~cjlin/liblinear` and the references therein.

Table 2.1. Data sets

Data set	number of data points	number of features
Gisette	6000	5000
RCV1	20242	47236
Alpha	500000	500
KDDA	8407752	20216830
KDDB	19264097	29890095
Epsilon	400000	2000
News20	19996	1355191
Synthetic	5000	5000
IC-Random	400000	100000
IC-Alpha	500000	500

A variety of methods has been proposed for solving problem (2.1), and high-performance implementations of some of these methods are available. One of the most popular codes is newGLMNET (Yuan et al., 2012), which is a C-implementation of a proximal Newton method and is a part of the LIBLINEAR package. Every iteration of this method identifies the active set as a subset of $\mathcal{A}^k$ as defined in (2.5), and then solves problem (2.2) inexactly using a randomized coordinate descent algorithm. The termination criterion for this inner loop is based on the ℓ_1-norm of the minimum norm subgradient and is adjusted by a heuristic as the iteration progresses.

We implemented Algorithm 2 in MATLAB, where we chose $\eta = 0.01$ and $\epsilon = 10^{-4}$. Subproblem (2.10) is solved inexactly via the conjugate gradient algorithm, which terminates as soon as the conjugate gradient iterate p satisfies

$$\frac{\|H^k p + g^k\|_\infty}{\|g^k\|_\infty} \leq 0.1.$$

Besides LIBLINEAR, we also compare our algorithm against FISTA (Beck and Teboulle, 2009) (part of the TFOCS package), OWL (Andrew and Gao, 2007) (implemented by

Schmidt (Schmidt, 2010)), and pdNCG (Fountoulakis and Gondzio, 2015). The latter implements a primal-dual Newton conjugate gradient method that replaces the ℓ_1-norm by a pseudo-Huber function. We include this solver because it was found to perform well on large and highly ill-conditioned problems; see (Fountoulakis and Gondzio, 2015).

Our comparisons are based on CPU time. We point out that most of the computational work in the FISTA, OBA, OWL, and pdNCG algorithms is spent in basic linear algebra operations (inner products and Matrix-vector products). As a consequence, the choice of programming language (MATLAB vs. C) has no significant impact on the computation time. In fact, we observed very similar performance of our own MATLAB and C implementations of OBA.

For all test problems, the regularization parameter μ was chosen through a 5–fold cross validation. LIBLINEAR, pdNCG and OBA use the exact Hessian in defining the quadratic model (2.10) and (2.2) while OWL uses a limited-memory BFGS approximation. Further, in order to solve singular problems, LIBLINEAR adds a small multiple (specifically, 10^{-12}) of the identity to the Hessian and our algorithm uses the value of 10^{-8}. LIBLINEAR employs a secondary mechanism to guard against singularity: it projects the result of the one dimensional optimization in the coordinate descent step onto the set $[-10, 10]$.

2.4.1. Test Results

The results of our numerical comparisons are presented in Figure 2.1. We plot the relative function error defined as

$$\frac{\phi(x^k) - \phi(x^\star)}{1 + \phi(x^\star)} \tag{2.15}$$

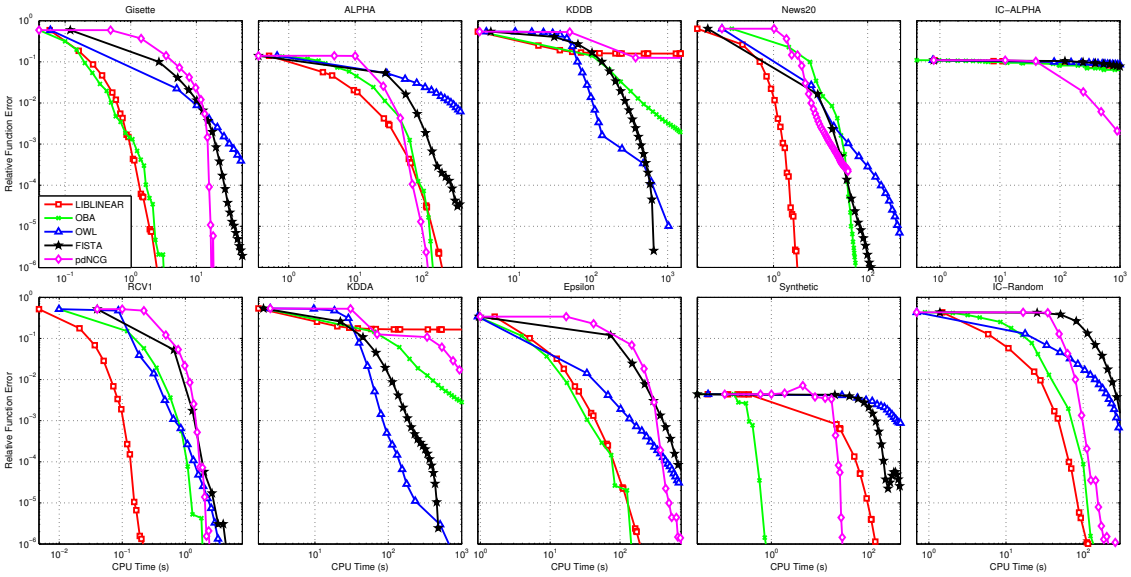


Figure 2.1. Relative error (2.15) in the objective function (vertical axis) versus CPU time

against CPU time. The value of $\phi(x^\star)$ was obtained by running our algorithm to a tight tolerance of 10^{-10} or until a time limit of 2000 CPU seconds was exceeded. The initial iterate for all methods was the zero vector.

It is interesting to compare LIBLINEAR and OBA. Both show strong performance overall, and there is no clear winner. This suggests that OBA is competitive with one the most efficient and popular solvers for logistic regression.

Further, we observe that for most of the data sets, FISTA is not competitive with the second-order methods. OWL has gained a reputation as an algorithm that performs well but unreliably so. The experiments support this opinion. For problems KDDA or KDDB, the performance of OWL is superior to the other solvers; however, for all other problems, OWL is not competitive due to poor steps and rapid changes in working orthant

faces. pdNCG is the only algorithm able to solve the (extremely ill conditioned) IC-Alpha problem in the required time-frame, but is not among the best solvers overall.

To better understand our method, Table 2.2 presents statistics regarding the number of the corrective loops (lines 10-15 of Algorithm 1). It can be seen that irrespective of the data set, less than 4 corrective loops are typically required per iteration. We emphasize that the strong performance of the proposed method is driven by the selective-corrective mechanism, not the ISTA backup, which was *never* invoked in these tests. Thus, the algorithm enjoys theoretical guarantees (unlike OWL) without compromising performance.

2.4.2. Sparsity

It is natural to ask whether an orthant-based method such as OBA is as effective at generating sparsity in the solution as a proximal Newton method, such as LIBLINEAR. In proximal Newton methods the non-smoothness of the original problem is retained in the subproblem (2.2), and sparsity arises because the solution of the subproblem typically lies at points of non-differentiability. In contrast, orthant-based methods solve a series of smooth problems that have no tendency of inducing sparsity in the solution by themselves, but achieve it through the projection of the trial point onto the working orthant.

Both methods, LIBLINEAR and OBA, also promote sparsity through the definition of the active set at the beginning of each (outer) iteration, but the construction of the active set differs in the two methods. LIBLINEAR fixes only a subset of the variables in the set $\mathcal{A}^k$ to zero; thus allowing some variables in $\mathcal{A}^k$ and all variables in the set $\mathcal{U}^k$ to move. On the other hand, OBA fixes all of the variables in $\mathcal{A}^k$ to zero and additionally fixes more variables in $\mathcal{U}^k$ through the selection mechanism and the corrective cycle. Therefore,

the approach in LIBLINEAR can be considered more liberal in that it releases more zero variables, while the approach in OBA can be regarded as more restrictive. Nevertheless, OBA becomes increasingly more liberal as the iteration progresses because the selection mechanism allows the size of the set $\mathcal{U}_F$ to double under certain circumstances (see step 17 of Algorithm 1).

In the light of these algorithmic differences, it is difficult to predict the relative ability of the two methods at generating sparse solutions. To explore this, we performed the following experiments using our data sets and recorded the sparsity in the solution. LIBLINEAR was used to solve the problems with the tolerance of their stopping criterion set to 10^{-3} (which yielded better misclassification rates than the default value of 10^{-2}), and OBA was then used to solve the problems to a similar accuracy in the objective function. The results are presented in Table 2.2 and show that the two methods achieve similar values of sparsity, with OBA being somewhat more effective. Results for the other three codes: OWL, LIBLINEAR and pdNCG, are not reported since they were not competitive in terms on sparsity on the data sets.

2.4.3. Subproblem Solver and Diagonal Dominance

Let us now focus on the methods used for solving the subproblems that incorporate second-order information about the objective function. It is natural to employ the conjugate gradient (CG) method in OBA, given that the subproblem (2.3) is smooth and that the CG method is an optimal Krylov process that can exploit problem structure effectively. An alternative to the CG method is the randomized coordinate descent algorithm, which has

Table 2.2. Solution Statistics. [a]Corresponds to number of iterations of the corrective loop (lines 10–15 of Algorithm 1). We report the 25^{th}, 50^{th} and 75^{th} quantile (Q_{25}, Q_{50}, and Q_{75}, resp.) and the maximum number of iterations of the loop.

	Correction Loops[a]				Percentage of Nonzeros in Solution	
Data set	Q_{25}	Q_{50}	Q_{75}	Maximum	LIBLINEAR	OBA
Gisette	3	3	3	4	88.92	**90.52**
RCV1	3.5	4	4	5	97.62	**97.65**
Alpha	1	2	3	3	**5.60**	5.20
KDDA	4	4	5	6	98.43	**98.71**
KDDB	4	4	5	6	97.11	**97.87**
Epsilon	1	2	2	3	44.60	**69.20**
News20	1	4	7	15	**99.60**	99.37
Synthetic	1	3	3	3	56.86	**58.82**
IC-Random	1.25	2	2.75	5	99.12	**99.13**
IC-Alpha	2	3	3	5	1.60	**9.00**

gained much popularity in recent years (Friedman et al., 2010; Nesterov, 2012; Richtárik and Takáč, 2014).

A drawback of coordinate descent for smooth unconstrained optimization is that it can be slow when the Hessian is not diagonally dominant. We experimented with a coordinate descent solver for the subproblem in OBA and found that its overall performance is inferior to that of the CG method.

The situation is quite different in a proximal Newton method where the subproblem is non-smooth. In that case, it is easy to compute the exact minimizer of (2.2) along each coordinate direction, thereby dealing explicitly with the non-differentiability of the original problem. Since this one dimensional minimization may return zero as the exact solution, the proximal coordinate descent method provides an active-set identification mechanism for the overall algorithm. Thus, although sensitivity to the lack of diagonal dominance may still be present, it is of a lesser concern due the benefits of its active set identification

Table 2.3. value of $\mathcal{D}(\nabla^2 f(x^0))$ as defined in (2.16)

Data set	$\mathcal{D}(\nabla^2 f(x^0))$
Gisette	57.99
RCV1	1.88
Alpha	9.93
KDDA	1.80
KDDB	1.50
Epsilon	5.55
News20	3.29
Synthetic	69.42
IC-Random	1.37
IC-Alpha	17.29

properties. Furthermore, applications in text classification and other areas often lead to problems with Hessians that are diagonally dominant (Greene and Cunningham, 2006).

This discussion motivates us to look more closely at the issue of diagonal dominance and its effect on the two methods. In order to quantify the level of diagonal dominance, we use the metric employed, for example, in (Wright, 2014). Given any symmetric matrix A, we define the level of diagonal dominance of A as

$$\mathcal{D}(A) = \frac{\max_i \|A_i\|_2}{\max_i |A_{ii}|}, \tag{2.16}$$

where A_i denotes the i^{th} column of A and A_{ii} denotes the i^{th} diagonal element of A. The smaller the value of $\mathcal{D}$, the closer is A to being diagonally dominant. In Table 2.3 we report the values of $\mathcal{D}\left(\nabla^2 f(x^0)\right)$ for all data sets in Table 2.1.

Let us begin by considering problem Synthetic, which was specifically constructed to have a high value of $\mathcal{D}$ (see Section 2.7 for details). Figure 2.1 shows that LIBLINEAR performs poorly compared to OBA. In additional experiments with variations of Synthetic, we observed that the relative performance of LIBLINEAR further deteriorates as $\mathcal{D}$

increases. The text classification tasks (RCV1 and News20), which are empirically observed to be diagonally dominant (Greene and Cunningham, 2006), have low values of $\mathcal{D}$ and, indeed, LIBLINEAR converges quickly[1]. However, LIBLINEAR also performs well on problem Gisette for which $\mathcal{D}$ is large and poorly on KDDB for which $\mathcal{D}$ is low. An examination of the rest of the results prevents us from establishing a clear correlation between the value of $\mathcal{D}$ and the relative performance of the two methods. Given the inconclusive nature of these results, we speculate that other factors such as solution sparsity, the conditioning of the subproblems, and the frequency of orthant changes are likely to contribute to the performance differences. We direct the reader to (Fountoulakis and Gondzio, 2015) for an investigation of some of those factors, particularly ill-conditioning, on the performance of first- and second-order methods.

2.5. Final Remarks

In this chapter, we presented a second-order algorithm for solving convex ℓ_1-regularized problems. At each iteration, the algorithm tries to predict the orthant face containing the solution, solves a smooth quadratic subproblem on this orthant face, and then invokes a corrective cycle that greatly improves the efficiency and robustness of the algorithm. We globalized the method by using the ISTA step as a reference for the desired progress. This enabled us to prove a linear convergence rate of the iterates for strongly convex problems. The ISTA backup is rarely used in practice (and never in the reported experiments) and thus, our theoretical result applies to a very robust method that invokes the safeguarding very rarely. This globalization procedure is analogous to a Newton trust-region method

[1]Interestingly, the LIBLINEAR website and manual convey that LIBLINEAR is known to perform well on document classification tasks but not necessarily on others.

where the underlying method is known to be very effective but convergence can only be proved by overcoming pathological situations with a first-order Cauchy step.

Numerical experiments for logistic regression show that our algorithm is competitive in terms of CPU time with state-of-the-art codes on a diverse collection of data sets. The algorithm is also effective in generating sparse solutions quickly. Overall, our experiments indicate that orthant-based methods are a viable alternative to proximal Newton methods.

2.6. Convergence Analysis

Recall that we wish to solve the problem

$$\min_{x \in \mathbb{R}^n} \ \phi(x) = f(x) + \mu \|x\|_1.$$

For the purpose of our analysis, we make two assumptions:

Assumption 2.6.1. *The function f is in C^1 and strongly convex with parameter $\lambda > 0$; i.e., for any $x, y \in \mathbb{R}^n$ and $t \in [0, 1]$:*

$$f(tx + (1-t)y) \leq tf(x) + (1-t)f(y) - \frac{1}{2}\lambda t(1-t)\|x - y\|_2^2. \tag{2.1}$$

As shown in Nesterov (2004), for continuously differentiable functions, this assumption is equivalent to

$$f(y) \geq f(x) + \nabla f(x)^T(y - x) + \frac{\lambda}{2}\|y - x\|_2^2 \qquad \text{for all } x, y \in \mathbb{R}^n. \tag{2.2}$$

Assumption 2.6.2. *The gradient of f is Lipschitz continuous with constant $L > 0$; i.e., for any $x, y \in \mathbb{R}^n$,*

$$\|\nabla f(x) - \nabla f(y)\|_2 \leq L\|x - y\|_2$$

The purpose of this section is to prove global convergence of the algorithm under the above assumptions. First, in order to guarantee that the algorithm is well-defined, we establish that the correction loop (lines 10-15 of Algorithm 1) and the line search (lines 20-22 of Algorithm 1) terminate in a finite number of iterations.

The finite termination of the correction loop follows trivially from the nature of the loop and the definition of $\mathcal{U}_F$. During each correction loop, elements are removed from $\mathcal{U}_F$ and added to $\mathcal{A}$. Note that at least one element must be removed from $\mathcal{U}_F$ during each corrective iteration. Since the loop begins with a finite number of elements in $\mathcal{U}_F$, one of two things can happen. Either (i) the loop terminates with $V = \phi$ but $\mathcal{U}_F \neq \phi$ or (ii) all elements in $\mathcal{U}_F$ are added to $\mathcal{A}$ causing $\mathcal{U}_F = \phi$ and consequently, $V = \phi$. In either case, the loop terminates in a finite number of iterations.

We now prove the finite termination of the line search.

Theorem 2.6.3. *The backtracking projected line search (steps 20–22 of Algorithm 1) terminates in a finite number of iterations.*

Proof. Consider the k^{th} iteration of Algorithm 1. For notational simplicity, we drop the iteration index and denote the iterate as x, the direction obtained after the corrective loop (steps 10–15) as d, and the smooth and non-smooth quadratic approximations as $\bar{q}$ and q, respectively. Along the same lines, let $\mathcal{A}$ and $\mathcal{U}$ be the active and unsure sets during this iteration.

We first show that there exists an $\bar{\alpha} > 0$ such that for any $\alpha > 0$ with $\alpha \leq \bar{\alpha}$, we have $\mathcal{P}(x + \alpha d) = x + \alpha d$.

Let $\mathcal{I}_1 = \{i \in \{1, 2, 3, \cdots, n\} : x_i \neq 0\}\}$ and $\mathcal{I}_2 = \{i \in \{1, 2, 3, \cdots, n\} : x_i = 0\}\}$, and let $\bar{\alpha} > 0$ such that $\bar{\alpha} < \left|\frac{x_i}{d_i}\right|$ for all $i \in \mathcal{I}_1$ with $d_i \neq 0$.

Let $\alpha > 0$ be such that $\alpha \leq \bar{\alpha}$ and ζ be defined in (2.12). We consider two cases:

- **Case 1:** $i \in \mathcal{I}_1$

  Let us concentrate on the case when $d_i \neq 0$. Notice that the case when $d_i = 0$ is trivial since irrespective of α, $\mathcal{P}(x_i + \alpha d_i) = \mathcal{P}(x_i) = x_i$.

  Now, assuming $d_i \neq 0$, because $\alpha \leq \bar{\alpha} < \left|\frac{x_i}{d_i}\right|$, it is clear that $\text{sgn}(x_i + \alpha d_i) = \text{sgn}(x_i) = \zeta_i$, and therefore $\mathcal{P}(x_i + \alpha d_i) = x_i + \alpha d_i$.

- **Case 2:** $i \in \mathcal{I}_2$

  By definition, $x_i = 0$. If $i \in \mathcal{A}$ in step 19 of Algorithm 1, then $d_i = 0$ and irrespective of ζ_i and α, $\mathcal{P}(x_i + \alpha d_i) = \mathcal{P}(0) = 0 = x_i + \alpha d_i$.

  Otherwise, $i \in \mathcal{U}_F \subseteq \mathcal{U}$, and therefore $\text{sgn}(-g_i) = \zeta_i \in \{-1, 1\}$. Assume that $\zeta_i = 1$. Thus, $\text{sgn}(-g_i) = 1$, and since $V^k = \emptyset$ and $i \in \mathcal{U}_F$, $\text{sgn}(x_i^k + d_i^k) = \text{sgn}(d_i^k) = 1$, so $d_i > 0$ which in turn implies $\alpha d_i > 0$. The same conclusion can be made if $\zeta_i = -1$. Thus, $\mathcal{P}(x_i + \alpha d_i) = \mathcal{P}(\alpha d_i) = \alpha d_i = x_i + \alpha d_i$.

Because d is a minimizer of $\bar{q}(x + d)$ in some subspace, we have $\bar{q}(x + \alpha d) \leq \bar{q}(x)$ for sufficiently small $\alpha \leq \bar{\alpha}$. Then, $\mathcal{P}(x + \alpha d) = x + \alpha d$, i.e., $x + \alpha d$ is in the same orthant as x, and therefore $q(x + \alpha d) = \bar{q}(x + \alpha d) \leq \bar{q}(x) = q(x)$. As a consequence, the termination condition in the while-loop is satisfied after a finite number of iterations.

□

We now show that by ensuring that ϕ at the new iterate is no larger than the majorizing function Γ^k, we can establish linear convergence.

Theorem 2.6.4. *Suppose that Assumptions 2.6.1 and 2.6.2 hold. Then, the iterates* $\{x^k\}$ *generated by Algorithm 2 converge to the optimal solution* $x^\star$ *of problem* (2.1) *at a linear rate.*

Proof. Consider the k^{th} iteration of Algorithm 2. For notational simplicity, let us drop the iteration index and denote the minimum norm subgradient as g, the Hessian approximation as H, and the iterate as x. Further, as is well known, the ISTA point x^k_{ISTA}, computed in step 5 of Algorithm 2, is the minimizer of a proximal approximation of $\phi(x)$,

$$x^k_{\text{ISTA}} = \arg\min_y f(x) + (y-x)^T\nabla f(x) + \frac{L}{2}\|y-x\|_2^2 + \mu\|y\|_1. \tag{2.3}$$

Because of Assumption 2.6.2, for any $z_1, z_2 \in \mathbb{R}^n$,

$$f(z_2) \leq f(z_1) + \nabla f(z_1)^T(z_2 - z_1) + \frac{L}{2}\|z_2 - z_1\|_2^2. \tag{2.4}$$

In particular, by setting $z_1 = x$, $z_2 = x^k_{\text{ISTA}}$, we get

$$\begin{aligned} \phi(x^k_{\text{ISTA}}) &= f(x^k_{\text{ISTA}}) + \mu\|x^k_{\text{ISTA}}\|_1 \\ &\leq f(x) + \nabla f(x)^T(x^k_{\text{ISTA}} - x) + \frac{L}{2}\|x^k_{\text{ISTA}} - x\|_2^2 + \mu\|x^k_{\text{ISTA}}\|_1 \equiv \Gamma^k. \end{aligned} \tag{2.5}$$

Let us denote the point obtained as a consequence of the globalization mechanism, which will be the new iterate, as x^+. This corresponds to x^{k+1} in step 14 of Algorithm 2. Realize that the loop in steps 8–13 of Algorithm 2 terminates finitely because once $\bar{\alpha}$ drops

to a value below ϵ, it is set to 0 and then $\phi(x^k_{\text{ISTA}} + \bar{\alpha}\bar{d}) = \phi(x^k_{\text{ISTA}})$ and the sufficiency condition (step 8 of Algorithm 2) is trivially satisfied by (2.5).

By design, our algorithm generates the point x^+ such that

$$\phi(x^+) \leq f(x) + \nabla f(x)^T(x^k_{\text{ISTA}} - x) + \frac{L}{2}\|x^k_{\text{ISTA}} - x\|^2 + \mu\|x^k_{\text{ISTA}}\|_1. \tag{2.6}$$

Combining this equation with the fact that x^k_{ISTA} is the minimizer in objective (2.3), we have for any $d \in \mathbb{R}^n$ and $y = x + \frac{\lambda}{L}d$ that

$$\begin{aligned}
\phi(x^+) &\leq f(x) + \nabla f(x)^T\left(\frac{\lambda}{L}d\right) + \frac{L}{2}\left\|\frac{\lambda}{L}d\right\|_2^2 + \mu\left\|x + \frac{\lambda}{L}d\right\|_1 \\
&\leq \phi\left(x + \frac{\lambda}{L}d\right) - \frac{\lambda}{2}\left\|\frac{\lambda}{L}d\right\|_2^2 + \frac{L}{2}\left\|\frac{\lambda}{L}d\right\|_2^2 \\
&= \phi\left(x + \frac{\lambda}{L}d\right) + \frac{\lambda^2}{2L}\left(1 - \frac{\lambda}{L}\right)\|d\|_2^2,
\end{aligned}$$

where the second inequality follows from (2.2) with $y = x + \frac{\lambda}{L}d$. In particular, we can set d to be $x^\star - x$ and obtain

$$\phi(x^+) \leq \phi\left(x + \frac{\lambda}{L}(x^\star - x)\right) + \frac{\lambda^2}{2L}\left(1 - \frac{\lambda}{L}\right)\|x^\star - x\|_2^2. \tag{2.7}$$

Using Assumption 2.6.1 and the convexity of the ℓ_1-norm, we have, for any $z_1, z_2 \in \mathbb{R}^n$ and $t \in [0, 1]$,

$$\phi(tz_1 + (1-t)z_2) \leq t\phi(z_1) + (1-t)\phi(z_2) - \frac{1}{2}\lambda t(1-t)\|z_1 - z_2\|_2^2.$$

Setting $z_1 = x$, $z_2 = x^\star$, and $t = \left(1 - \frac{\lambda}{L}\right)$, we get

$$\phi\left(x + \frac{\lambda}{L}(x^\star - x)\right) \leq \frac{\lambda}{L}\phi(x^\star) + \left(1 - \frac{\lambda}{L}\right)\phi(x) - \frac{\lambda^2}{2L}\left(1 - \frac{\lambda}{L}\right)\|x^\star - x\|_2^2.$$

Combining this result with (2.7), we get

$$\begin{aligned}\phi(x^+) &\leq \frac{\lambda}{L}\phi(x^\star) + \left(1 - \frac{\lambda}{L}\right)\phi(x) - \frac{\lambda^2}{2L}\left(1 - \frac{\lambda}{L}\right)\|x^\star - x\|_2^2 + \frac{\lambda^2}{2L}\left(1 - \frac{\lambda}{L}\right)\|x^\star - x\|_2^2 \\ &= \phi(x^\star) + \left(1 - \frac{\lambda}{L}\right)(\phi(x) - \phi(x^\star)),\end{aligned}$$

and therefore

$$\phi(x^+) - \phi(x^\star) \leq \left(1 - \frac{\lambda}{L}\right)(\phi(x) - \phi(x^\star)).$$

By reintroducing the iteration index and using recursion, we have that

$$\phi(x^k) - \phi(x^\star) \leq \left(1 - \frac{\lambda}{L}\right)^k (\phi(x^0) - \phi(x^\star))$$

as required.

□

2.7. Reproducible Research

The MATLAB code used to generate the "Synthetic" problem is presented below. Given a dimension `n`, we use the following code snippet to generate the vector of labels (denoted by `y`) and the data matrix (denoted by `X`).

```
y=-1+(rand(n,1)>0.5)*2;

X = rand(n,n);

X = X + X';
```

```
mineig = min(eig(X));

if(mineig<0)

 X = eye(size(X))*mineig*-2+X;

end

X = chol(X);
```

The code for our method can be found at `https://github.com/keskarnitish/OBA`.

CHAPTER 3

A Limited-Memory Quasi-Newton Algorithm for Bound-Constrained Nonsmooth Optimization

3.1. Introduction

We propose an algorithm for solving bound-constrained optimization problems of the form

$$\min_{x \in \mathbb{R}^n} f(x) \quad \text{s.t. } l \leq x \leq u, \tag{3.1}$$

where the objective function $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ is continuous but might not be differentiable everywhere. The lower bounds $l \in (\mathbb{R} \cup \{-\infty\})^n$ and upper bounds $u \in (\mathbb{R} \cup \{\infty\})^n$ can take values of $-\infty$ or ∞ whenever the variables are unbounded in those coordinates. We assume that the problem is feasible, i.e., $l \leq u$, and allow for f to be nonconvex.

Many algorithms have been proposed for solving (3.1) when x is unconstrained. Some of these methods include gradient-sampling methods (Burke et al., 2005; Curtis and Que, 2013; Kiwiel, 2007), bundle methods (Haarala et al., 2004, 2007; Mäkelä, 2002), quasi-Newton methods (Curtis and Que, 2015; Kaku, 2011; Lewis and Overton, 2013; Lewis and Zhang, 2015; Skajaa, 2010), and hybrid methods (Curtis and Que, 2015). Gradient-sampling methods randomly sample gradients in the vicinity of the iterate

to calculate an estimate of the minimum-norm subgradient. In conjunction with an Armijo-like line search, global convergence can be proved using these minimum-norm subgradients as search directions. Bundle methods aggregate subgradients from previous iterates and iteratively solve piecewise-quadratic approximations of the objective function to generate steps. Recently, Lewis and Overton (Lewis and Overton, 2013) observed that the unadulterated BFGS method works very well when applied to unconstrained nonsmooth optimization problems so long as the weak Wolfe line search is performed. Skajaa (Skajaa, 2010) reported similar results for L-BFGS (Liu and Nocedal, 1989). For problems ranging from $n = 100$ to $n = 10000$, it was found that L-BFGS was not only more efficient in solving test problems, but it was also more reliable compared to other methods. However, theoretical convergence guarantees (or the lack thereof) of (L-)BFGS for nonsmooth problems remain to be shown. Recent efforts (e.g., (Curtis and Que, 2015)) focused on the design of a hybrid strategy that retains the efficacy of standard L-BFGS but ensures convergence through a gradient-sampling approach. Other approaches for solving (3.1) include subgradient methods, quasi-secant methods, and discrete gradient methods. We refer the reader to (Bagirov et al., 2014; Clarke, 1990; Karmitsa et al., 2012; Mäkelä, 2002; Skajaa, 2010) for a detailed summary of these methods and their numerical performance.

In certain applications, it is necessary to optimize a nonsmooth objective function subject to bound constraints. These include applications in many fields including statistics, optimal control, and as subproblems for certain robust optimization problems (Bagirov et al., 2014). Some of the algorithms described above can be extended to solve problems with bound constraints. For instance, the LMBM-B (Karmitsa and Mäkelä, 2010a,b)

method extends the limited-memory bundle method to (3.1). Gradient-sampling methods have also been extended to the case of constrained optimization (Curtis and Overton, 2012). A natural question is whether the surprising and remarkable success of the unadulterated (L-)BFGS method in the unconstrained case can be extended to problems with bound constraints.

The L-BFGS-B method is a variant of L-BFGS for minimizing a smooth objective function over box constraints. At an iterate x^k, the method first determines an active set by computing a Cauchy point $\tilde{x}^k$ as the first local minimizer $\alpha > 0$ of f along the gradient-projection path $\alpha \mapsto P(x^k - \alpha \nabla f(x^k))$. Here, $P(v)$ is the orthogonal projection of a vector $v \in \mathbb{R}^n$ onto the feasible hypercube $[l, u]$. The bound constraints that are tight at the Cauchy point $\tilde{x}^k$ then define an active set, $\mathcal{A}^k = \{i : \tilde{x}_i^k \in \{l_i, u_i\}\}$, and a subspace step $\tilde{p}^k$ is computed as the solution of the problem

$$\begin{aligned} \min_{p \in \mathbb{R}^n} \; & f(x^k) + \nabla f(x^k)^T p + \frac{1}{2} p^T B^k p \\ \text{s.t. } & p_i = 0 \text{ for all } i \in \mathcal{A}^k. \end{aligned} \tag{3.2}$$

The objective in this subproblem is a quadratic model of the original objective function. Its Hessian matrix B^k is defined by the L-BFGS update and therefore is positive definite. Subproblem (3.2) is solved efficiently using the compact-form representation of L-BFGS (Byrd et al., 1994b). Finally, the overall step $p^k = (\tilde{x}^k + \tilde{p}^k) - x^k$ is computed and a projected line search is performed along the path $\alpha \mapsto P(x^k + \alpha p^k)$ to find a step size satisfying the strong Wolfe conditions.

Henao et al. (Henao, 2014) recently proposed L-BFGS-B-NS as a variant of L-BFGS-B for solving (3.1) with a nonsmooth objective function. The only difference to the original method is that the strong Wolfe line search is replaced with the weak Wolfe line search. This is the same modification that was suggested by Lewis and Overton (Lewis and Overton, 2013) in the unconstrained case.

In this paper, we propose a different adaptation of the L-BFGS method. Our method first determines an active set based on the bound constraints that are tight at the current iterate, without referring to a Cauchy point. After computing the search direction from (3.2), a new iterate is determined using a variant of the weak Wolfe line search.

The key ingredients in our method are active-set selection strategies that take into account the nonsmoothness of the function. First, we propose the use of an approximation of the minimum-norm ϵ-subgradient instead of the gradient to determine which bound constraints are binding at the current iterate. Second, we explore an iterative corrective mechanism that augments the active set until the final search direction points inside the feasible region.

Throughout the paper, we assume that the function is differentiable at each iterate and trial point, and that its gradient can be computed. This is in line with the work by Lewis and Overton (Lewis and Overton, 2013) who make the same assumption for their numerical experiments. Burke et al. (Burke et al., 2005) describe a mechanism that perturbs a trial point in case f is not differentiable at that point. In the box-constrained case, the boundary of the feasible region might align with a manifold of nondifferentiability. Then, the projections carried out during the line search might generate trial points that lie in this manifold. To circumvent this problem, we assume that there is an extension

of the function beyond the feasible region that is differentiable at almost all boundary points. For example, consider the feasible set $[0, 1] \subset \mathbb{R}$ with objective function $f(x) = |x|$ which is not differentiable at the boundary point $x = 0$. When we replace the objective with $\tilde{f}(x) = x$, the objective values are identical within the feasible region, resulting in the same optimal solution, but the function is now differentiable at $x = 0$.

The paper is organized as follows. In the subsection to follow, we introduce some notation that is used throughout the paper. In Section 2, we describe our algorithm including the active-set prediction and correction strategies, as well as the proposed weak Wolfe line search. Finally, in Section 3, we present details of our Python implementation and detailed numerical results examining the efficacy of our approach.

3.1.1. Notation

We use superscripts to denote the iteration index and subscripts to denote a specific element of a vector. For instance, x_j^k refers to the j^{th} element of the k^{th} iterate. We abbreviate $[\nabla f(x)]_i$ by $\nabla_i f(x)$. We define the instantaneous projection of a direction p at an iterate x as

$$[T(x,p)]_i = \begin{cases} p_i & \text{if } x_i \in (l_i, u_i) \\ \max(p_i, 0) & \text{if } x_i = l_i \\ \min(p_i, 0) & \text{if } x_i = u_i. \end{cases} \tag{3.3}$$

This operator zeroes out those components of p for which x is at its bounds with p pointing in the direction of infeasibility. We use the notation $B_\epsilon(x)$ to denote the closed ℓ_2-norm ball of radius ϵ centered at x. Further, we denote the cardinality of a set A by $|A|$. Finally,

given a vector v and a set of indices $\mathcal{A}$, $v_{\mathcal{A}}$ refers to the subvector corresponding to the indices in $\mathcal{A}$. Similarly, given a matrix M, then $M_{\mathcal{A},\mathcal{B}}$ denotes the submatrix with row indices given in $\mathcal{A}$ and column indices given in $\mathcal{B}$. Finally, we let $\mathcal{N} = \{1, \ldots, n\}$ be the set of all variable indices.

3.2. Proposed Algorithm

3.2.1. Active-Set Framework

The proposed algorithm is an active-set method which, at each iteration, determines an estimate $\mathcal{A}^k$ of the optimal active set $\mathcal{A}^* := \{i \in \mathcal{N} : x_i^* = l_i \text{ or } x_i^* = u_i\}$ of a local solution x^* of (3.1). We say that the bound constraints in $\mathcal{A}^*$ are tight at x^*. The L-BFGS-B algorithm chooses as active set $\mathcal{A}^k$ the bounds at which the Cauchy point are tight. In contrast, our method chooses from bounds for which the current iterate x^k itself is tight, without referring to a Cauchy point.

For a smooth objective function, we might consider the set of the tight constraints that are binding; i.e.,

$$\mathcal{B}^k(g^k) = \{i \in \mathcal{N} : x_i^k = l_i \text{ and } g_i^k \geq 0\} \cup \{i \in \mathcal{N} : x_i^k = u_i \text{ and } g_i^k \leq 0\} \tag{3.4}$$

with $g^k = \nabla f(x^k)$. These are the coordinates for which the gradient predicts no decrease in the objective if the corresponding components of the iterate are moved inside the feasible region. With this, $\nabla_i f(x^k) = 0$ for all $i \in \mathcal{N} \setminus \mathcal{B}(\nabla f(x^k))$ if and only if x^k satisfies the

first-order optimality conditions for problem (3.1) at x^k; i.e.,

$$\begin{aligned} &\nabla_i f(x^k) = 0 \text{ for all } i \text{ with } l_i < x_i^k < u_i \\ &\nabla_i f(x^k) \geq 0 \text{ for all } i \text{ with } x_i^k = l_i \\ &\nabla_i f(x^k) \leq 0 \text{ for all } i \text{ with } x_i^k = u_i. \end{aligned} \tag{3.5}$$

Consequently, the subspace step p^k obtained from solving (3.2) with $\mathcal{A}^k = \mathcal{B}(\nabla f(x^k))$ is zero if and only if x^k is a first-order optimal point. In addition, it can be shown that p^k is a descent direction for the projected line search; i.e., the function $\alpha \mapsto f(P(x^k + \alpha p^k))$ is decreasing for $\alpha > 0$ sufficiently small. (This is a consequence of (Bertsekas, 1982, Proposition 1).)

Consider a simple algorithm that computes search directions from (3.2) with $\mathcal{A}^k = \mathcal{B}(\nabla f(x^k))$ and performs a projected line search to determine the new iterate $x^{k+1} = P(x^k + \alpha^k p^k)$ with some step size $\alpha^k > 0$. Suppose that f is differentiable and that the iterates converge to a non-degenerate first-order optimal x^*; i.e., x^* satisfies (3.5) and $\nabla_i f(x^*) \neq 0$ for all $i \in \mathcal{A}^*$. Further assume that at some iterate x^k sufficiently close to x^*, the bounds that are tight at x^k are identical to the optimal active set; i.e., $\{i \in \mathcal{N} : x_i^k = l_i \text{ or } x_i^k = u_i\} = \mathcal{A}^*$. It is then not difficult to show that $\mathcal{A}^k = \mathcal{B}(\nabla f(x^*))$ for all large k. In other words, the optimal active set is identified in a finite number of iterations. This observation motivates the choice $\mathcal{A}^k = \mathcal{B}(\nabla f(x^k))$.

The conclusion in the previous paragraph was drawn under the strong assumption that an iterate is encountered at which all constraints in $\mathcal{A}^*$ are tight. To the best of our knowledge, no convergence proof has been established for the simple algorithm when

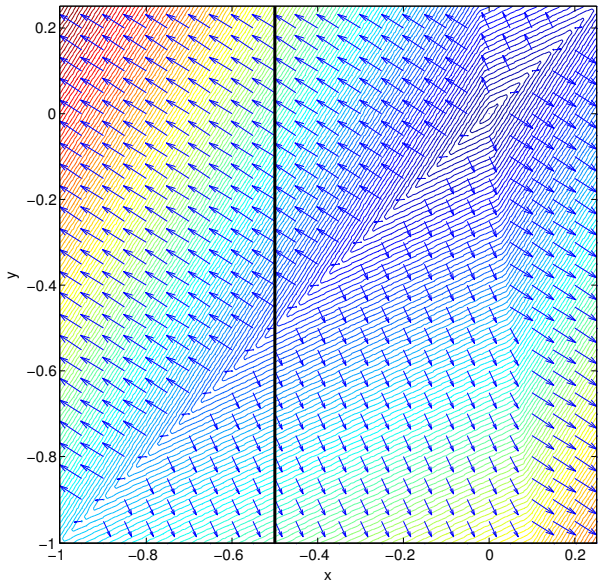


Figure 3.1. The contour lines of the objective function in (3.6). The arrows indicate the gradients of the function.

this assumption is lifted, even when f is differentiable. Nevertheless, despite the lack of theoretical convergence guarantees, our proposed active-set selection strategy is based on (3.4) since it seems to perform well in practice in our setting. Recall that global convergence has not been proved for the unadulterated L-BFGS algorithm with a nonsmooth objective function even in the unconstrained case.

In the context of nonsmooth optimization, the gradient of the objective function can be very myopic in regions close to a manifold on which the function is nondifferentiable, and a gradient-based active-set identification can be quite misleading. We illustrate this with a simple example, depicted in Figure 3.1.

Consider the problem

$$\min_{x\in\mathbb{R}^2} |x_1 - x_2| + \frac{1}{2}(x_1 + 0.1x_2)^2 \quad \text{s.t. } x_1 \leq -0.5 \tag{3.6}$$

with optimal solution $x^* = (-0.5, -0.5)^T$. Suppose we have an iterate $x^k = (-0.5, a)^T$ for a number $a \in (-5, -0.5)$. The function is differentiable at this point with gradient $\nabla f(x^k) = (1, -1)^T + (-0.5 + 0.1a)(1, 0.1)^T = (0.5 + 0.1a, -1.05 + 0.01a)^T$. Given that $a > -5$, we have $\nabla_1 f(x^k) > 0$. Therefore, the choice $\mathcal{A}_k = \mathcal{B}(\nabla f(x^k))$ predicts x_1 to be free, no matter how close x^k is to x^*. This determination is incorrect since x_1 is indeed at its bound at the solution. Notice that the failure of identification is caused by the inherent shortsightedness of the gradient and not due to some kind of degeneracy.

We point out that also the active-set identification of the L-BFGS-B method does not recognize that the bound for x_1 is tight at the solution. The L-BFGS-B method, as described previously, locates the first minimizer of the gradient-projection path, $\alpha \mapsto P(x^k - \alpha\nabla f(x^k))$, for the active-set identification. For problem (3.6), the ray $\{x^k - \alpha\nabla f(x^k) : \alpha > 0\}$, for an iterate $x^k = (-0.5, a)^T$ with $a \in (-5, -0.5)$, never intersects a bound. Thus, also in this case, x_1 is not recognized as active.

In order to compensate for the shortsightedness of the gradient, we propose two strategies: (i) an active-set prediction that considers an approximation $\tilde{g}^k$ of minimum-norm subgradients of nearby non-differentiable points to determine the binding constraints $\mathcal{B}(\tilde{g}^k)$ (Section 3.2.2); and (ii) a correction mechanism that augments the active set if

the search direction, computed with L-BFGS approximation of the nonsmooth objective, indicates that a variable should be active (Section 3.2.4).

3.2.2. Active-Set Prediction Using a Subgradient Approximation

Continuing with problem (3.6), let us consider the scenario in which we use the ϵ-minimum-norm subgradient (ϵ-MNSG) based on the Clarke ϵ-subdifferential (Clarke, 1990) for the active-set prediction. For a fixed $\epsilon > 0$, the ϵ-MNSG is defined as

$$\hat{g}_\epsilon(x) = \underset{y \in \text{cl conv } \nabla f(B_\epsilon(x))}{\arg\min} \quad \frac{1}{2}\|y\|_2^2. \tag{3.7}$$

Here, $\nabla f(B_\epsilon(x)) = \{\nabla f(y) : y \in B_\epsilon(x)\}$, and the term "cl conv" indicates the closure of the convex hull of a set. When ϵ is sufficiently small and a is close to -0.5, the ϵ-MNSG at $x^k = (-0.5, a)^T$ is approximately $\hat{g}_\epsilon(x^k) \approx (-0.3025, -0.3025)^T$. The active set $\mathcal{A}^k = \mathcal{B}(\hat{g}_\epsilon(x^k))$ correctly identifies that x_1 is tight at the solution. Indeed, the ϵ-MNSG attempts to be less myopic than the gradient and forms the basis for gradient-sampling methods (Burke et al., 2005). This motivates us to base the active-set identification on the ϵ-MNSG instead of the gradient.

Computing the true ϵ-MNSG is usually not feasible, due to the complex nature of $\nabla f(B_\epsilon(x^k))$. Instead, gradient sampling methods work with an approximation $\tilde{g}^k$ that is based on the gradients at points from a finite random subsample $G^k = \{x^{k,1}, \ldots, x^{k,l^k}\}$ of the ball $B_\epsilon(x^k)$ with $x^k \in G^k$. More specifically, $\tilde{g}^k = \sum_{i=1}^{l^k} \nabla f(x^{k,i})\lambda_i^\star$ where $\lambda^\star$ is the

solution of the convex quadratic problem

$$\begin{aligned} \min_{\lambda \in \mathbb{R}^{l^k}} \ & \frac{1}{2} \left\| \sum_{i=1}^{l^k} \nabla f(x^{k,i}) \lambda_i \right\|_2^2 \\ \text{s.t.} \ & \sum_{i=1}^{l^k} \lambda_i = 1 \\ & 0 \leq \lambda_i \leq 1 \quad \text{for all } i = 1, \ldots, l^k. \end{aligned} \tag{3.8}$$

A good approximation of the ϵ-MNSG typically requires a large number l^k of gradient evaluations. For the purpose of determining the active set, however, an inexact estimate might suffice, since its main purpose is to capture roughly the geometry of the nonsmooth function. It is not used for the step computation itself. To avoid additional gradient evaluations, we simply choose $G^k = \{x^k, \ldots, x^{\max\{0,k-M\}}\}$ to contain the most recent M iterates. This strategy is motivated by two observations: (i) as we will describe in Section 3.2.5, the line search encourages the iterates to cross over manifolds of nondifferentiability and thus, the gradients for G^k represent different "pieces" of the nonsmooth function; and (ii) near the solution, where active-set prediction strategies are arguably more important, the steps taken by the algorithm are small and the points in G^k are then from a small neighborhood around the current iterate.

Motivated by these observations, our first active-set selection strategy chooses

$$\mathcal{A}^k = \mathcal{B}^k(\tilde{g}^k) \cup \mathcal{B}^k(\nabla f(x^k)). \tag{3.9}$$

Note that we include the bound constraints identified by the gradient $\nabla f(x^k)$ as well. We observed in our experiments that this led to better performance. We speculate that,

in regions not close to a manifold on which the function is nonsmooth, the active set identified by the gradient is often reliable and the subgradient approximation might cause spurious identification, when the points in G^k are not close to each other.

3.2.3. Computation of the Search Direction

Our second active-set strategy loops over candidate choices for the active set that are evaluated based on the search directions they generate. We describe the step computation first.

The search directions are based on the BFGS method (Nocedal and Wright, 2006). This method constructs and updates a convex second-order model of the objective function requiring only the first-order derivatives. Given an estimate, B^l, of the curvature of the objective function, the BFGS method revises the estimate using a rank-2 update as

$$B^{l+1} = B^l + \frac{y^l (y^l)^T}{(y^l)^T s^l} - \frac{B^l s^l (s^l)^T B^l}{(s^l)^T B^l s^l}, \tag{3.10}$$

where

$$s^l = x^{l+1} - x^l \tag{3.11a}$$

$$y^l = \nabla f(x^{l+1}) - \nabla f(x^l). \tag{3.11b}$$

One usually requires that the curvature condition

$$(s^l)^T y^l > 0 \tag{3.12}$$

holds, since then B^l remains positive definite if the initial matrix B^0 is positive definite. For smooth convex objective functions, the BFGS method possesses strong theoretical properties including global convergence and superlinear local convergence. Even though only limited theoretical convergence guarantees have been established for nonconvex objectives, many have noted good performance on a variety of problems.

The original BFGS method requires the storage and manipulation of an $n \times n$ matrix. For large-scale problems, this is unwieldy. The limited-memory BFGS (L-BFGS) method (Liu and Nocedal, 1989) attempts to alleviate this handicap by storing only the past m curvature pairs (s^l, y^l). The matrix B^l itself is never explicitly constructed. The value for m, also called the L-BFGS memory, is often in the range of 5–20. This reduces the storage from $\mathcal{O}(n^2)$ to $\mathcal{O}(mn)$. The complexity of the search direction computation in the L-BFGS method is also reduced from $\mathcal{O}(n^2)$ to $\mathcal{O}(mn)$.

Given an active set $\mathcal{A}^k$ and an L-BFGS approximation B^k of the Hessian of the objective function, we generate the search direction p^k as the solution to subproblem (3.2). Here, all components of p^k belonging to $\mathcal{A}^k$ are set to zero, and the remaining entries are obtained from a linear system involving a symmetric submatrix of B^k. Making use of the L-BFGS compact representation matrices (Byrd et al., 1995, 1994b), the step can be computed efficiently, using $2m^2t + 6mt + 4t + \mathcal{O}(m^3)$ operations where $t = |\mathcal{A}^k|$.

To specify the L-BFGS approximation in a given iteration k, it is necessary to provide an initial matrix $B^{k,0}$, from which $B^k = B^{k,m}$ is generated by repeatedly applying the update formula (3.10). The matrix $B^{k,0}$ is an estimate of the Hessian of f. This choice, especially when the L-BFGS memory m is low, has direct consequences on the quality of the search direction. For smooth optimization, $B^{k,0} = \theta I$ where $\theta = \frac{(s^k)^T y^k}{(y^k)^T y^k}$ is often recommended

and is found to work well in a variety of applications. Intuitively, the ratio is justified since it is a scalar approximation to $\nabla^2 f(x^k)$ (Nocedal and Wright, 2006). However, for nonsmooth optimization this choice seems to lead to inferior performance. Instead, Curtis and Que (Curtis and Que, 2015) proposed $\theta = \max(1.0, \min(\|\nabla f(x^k)\|_\infty, 10^8))$. We use this choice in our implementation as well.

3.2.4. An Active-Set Correction Mechanism

In Section 3.2.2, we described an active-set identification mechanism that is based on an approximation of a subgradient. This strategy attempts to guess directly which bounds are tight at the optimal solution.

Next we describe another approach using an iterative correction procedure that judges the quality of a candidate active set and adjusts it if necessary. The quality of the active set is judged through the search direction generated using it. The goal is to obtain a direction that is feasible in the sense that a sufficiently small step into this direction does not leave the feasible region. If, for a given candidate active set, there is a variable that is tight at the current iterate and the candidate search direction points outside the feasible region, then this variable is added to the active set and the procedure is repeated. Similar mechanisms have been used previously, for example, for solving convex quadratic programs (Curtis et al., 2015; Hungerländer and Rendl, 2015) and ℓ_1-regularized convex optimization problems (Byrd et al., 2016; Keskar et al., 2016).

For ease of notation, we drop the iteration index k for the remainder of this section. Given an iterate x, we let $g = \nabla f(x)$, and we define the set of interior variables, $\mathcal{F} = \{i : l_i < x_i < u_i\}$ and the set of variables with tight bound constraints, $\bar{\mathcal{F}} = \mathcal{N} \setminus \mathcal{F}$.

Algorithm 3 formally states the proposed active-set correction procedure. When there are no tight bounds, the active set must be empty, and step 4 immediately returns the unrestricted search direction in the full space. Otherwise, the t-loop computes a potential search direction in step 7 and tests if there are any components, collected in the set $\mathcal{C}^t$, that would take a variable instantaneously outside its bound constraints. Such components are added to the active set, until a feasible direction is found. Clearly, the loop terminates in finite time, since $\mathcal{A}^t$ grows by at least one element per iteration.

Algorithm 3 ActiveSetCorrection

Inputs: Current iterate x; initial active set $\mathcal{A}^{\text{init}} \subseteq \bar{\mathcal{F}}$.

Output: Final active set $\mathcal{A}$ with corresponding search direction p.

1: Initialize $t \leftarrow 0$ and set $\mathcal{A}^0 = \mathcal{A}^{\text{init}}$.
2: **if** $\bar{\mathcal{F}} = \emptyset$ **then** ▷ Nothing to do if there are no tight bounds
3: Compute p as solution of (3.2) with $\mathcal{A}^k = \emptyset$.
4: **return** $\mathcal{A} = \emptyset$ and p.
5: **end if**
6: **for** $t = 0, 1, 2, \ldots$ **do**
7: Compute p^t as solution of (3.2) with $\mathcal{A}^k = \mathcal{A}^t$. ▷ Potential search direction
8: Set $\mathcal{C}^t = \{i \in \bar{\mathcal{F}} \setminus \mathcal{A}^t : T(x, p^t)_i \neq p_i^t\}$. ▷ Variables to be added
9: **if** $\mathcal{C}^t = \emptyset$ **then**
10: **return** $\mathcal{A} = \mathcal{A}^t$ and $p = p^t$. ▷ No more corrections necessary
11: **end if**
12: Set $\mathcal{A}^{t+1} = \mathcal{A}^t \cup \mathcal{C}^t$.
13: **end for**

The approach described in Section 3.2.2 uses a subgradient approximation to overcome the shortsightedness of the gradient when predicting the optimal active set. In contrast,

the corrective procedure in Algorithm 3 exploits the fact that the L-BFGS approximation of the problem curvature contains information about the structure of the nonsmoothness. It has been observed that the (L-)BFGS approximation of a nonsmooth objective function is able to approximate the U- and V-spaces of the objective function (Lewis and Overton, 2013; Skajaa, 2010). Roughly speaking, the U-space of f at a point x is the subspace tangent to the manifold of points at which f is not differentiable. The V-space is the orthogonal complement of the U-space. In (Lewis and Overton, 2013), Lewis and Overton hypothesized that the V-space of a nonsmooth function can be numerically approximated within the unadulterated BFGS method through the eigenvectors of B^k corresponding to eigenvalues that converge to infinity.

In our example problem (3.6), the V-space at any point x with $x_1 = x_2$, including the optimal solution, is spanned by $(1, -1)^T$. When we apply the proposed method from random starting points, the iterates converge to the solution $(-0.5, -0.5)^T$. After some iterations, the BFGS matrix is approximately

$$B^k \approx y^k \cdot \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \tag{3.13}$$

with some sequence y^k converging to infinity. Note that the eigenvectors of B^k suggest precise recovery of the U- and V-spaces of the objective function. In particular, the eigenvector $(1, -1)^T$ with respect to the asymptotically infinite eigenvalue indeed spans the V-space of f at the optimal solution. Even though the matrix on the right-hand side of (3.13) is singular, B^k itself is always nonsingular. Numerically we observe that the

inverse matrix $H^k = (B^k)^{-1}$ is approximately

$$H^k \approx \tilde{y}^k \cdot \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \tag{3.14}$$

with some sequence $\tilde{y}^k$ converging to zero, and now $(1, -1)^T$ is an eigenvector with respect to the eigenvalue approaching zero.

Dropping the iteration index k, consider again an iterate of the form $x = (-0.5, a)^T$ with $a \in (-5, -0.5)$ and gradient $\nabla f(x) = (0.5 + 0.1a, -1.05 + 0.01a)^T$. In Section 3.2.1 we observed that the naïve choice $\mathcal{A} = \mathcal{B}(\nabla f(x)) = \emptyset$ fails to recognize that x_1 is active at the solution. If we choose $\mathcal{A}^{\text{init}} = \mathcal{B}(\nabla f(x))$ in Algorithm 3, we have $\mathcal{A}^0 = \emptyset$ in the first iteration. With the approximation (3.14), the search direction in step 7 becomes $p^0 \approx \tilde{y} \cdot (0.55 - 0.11a, 0.55 - 0.11a)^T$. Clearly, from the current iterate with $x_1 = -0.5$, this direction points out of the feasible region because $p_1^0 > 1$. Therefore, $T(x, p^0)_1 \neq p_1^0$ and $\mathcal{C}^0 = \{1\}$. In the next iteration of the correction loop, $\mathcal{A}^1 = \{1\}$ is accepted as the final active set, and correctly predicts that x_1 is active at the solution.

The following lemma shows that the correction mechanism cannot lead to a spurious termination of the overall algorithm. A zero step can be generated only when the current iterate is already a stationary point of the objective function (assuming that $\mathcal{A}^0$ is initialized as $\mathcal{B}(\nabla f(x^k))$).

Lemma 3.2.1. *Suppose $\mathcal{A}^0 = \mathcal{B}(\nabla f(x^k))$ and the corrective loop terminates with $p = 0$. Then, the current iterate x^k satisfies the first order optimality conditions* (3.5) *for problem* (3.1).

PROOF OF LEMMA 3.2.1. For ease of exposition, we assume that $l = 0$ and $u = \infty$ in problem (3.1). Let $\hat{t}$ be the iteration in which the method terminates with $p = p^{\hat{t}} = 0$ and let $g = \nabla f(x)$.

Given the active set $\mathcal{A}^{\hat{t}} \subseteq \bar{\mathcal{F}}$ and defining $\bar{\mathcal{A}}^{\hat{t}} := \bar{\mathcal{F}} \setminus \mathcal{A}^{\hat{t}}$, the solution p to (3.2) (with $\mathcal{A}^k = \mathcal{A}^{\hat{t}}$) is obtained by solving the linear system

$$\begin{bmatrix} B_{\mathcal{F}\mathcal{F}} & B_{\mathcal{F}\bar{\mathcal{A}}^{\hat{t}}} \\ B_{\bar{\mathcal{A}}^{\hat{t}}\mathcal{F}} & B_{\bar{\mathcal{A}}^{\hat{t}}\bar{\mathcal{A}}^{\hat{t}}} \end{bmatrix} \begin{pmatrix} p_{\mathcal{F}} \\ p_{\bar{\mathcal{A}}^{\hat{t}}} \end{pmatrix} = - \begin{pmatrix} g_{\mathcal{F}} \\ g_{\bar{\mathcal{A}}^{\hat{t}}} \end{pmatrix} \tag{3.15}$$

and setting

$$p_{\mathcal{A}^{\hat{t}}} = 0. \tag{3.16}$$

Because the L-BFGS approximation B is positive definite, (3.15) together with $p^{\hat{t}} = 0$ implies

$$g_{\mathcal{F}} = 0 \text{ and } g_{\bar{\mathcal{A}}^{\hat{t}}} = 0. \tag{3.17}$$

For the purpose of deriving a contradiction suppose that $\hat{t} > 0$. Then $\mathcal{A}^{\hat{t}} = \mathcal{A}^{\hat{t}-1} \cup \mathcal{C}^{\hat{t}-1}$, and therefore $\bar{\mathcal{A}}^{\hat{t}-1} = \bar{\mathcal{A}}^{\hat{t}} \cup \mathcal{C}^{\hat{t}-1}$. By definition, $p_i^{\hat{t}-1} < 0$ for every $i \in \mathcal{C}^{\hat{t}-1}$. Since $\mathcal{C}^{\hat{t}-1} \neq \emptyset$, we have $p^{\hat{t}-1} \neq 0$. Consider the linear system from which $p^{\hat{t}-1}$ is computed:

$$\underbrace{\begin{bmatrix} B_{\mathcal{F}\mathcal{F}} & B_{\mathcal{F}\bar{\mathcal{A}}^{\hat{t}}} & B_{\mathcal{F}\mathcal{C}^{\hat{t}-1}} \\ B_{\bar{\mathcal{A}}^{\hat{t}}\mathcal{F}} & B_{\bar{\mathcal{A}}^{\hat{t}}\bar{\mathcal{A}}^{\hat{t}}} & B_{\bar{\mathcal{A}}^{\hat{t}}\mathcal{C}^{\hat{t}-1}} \\ B_{\mathcal{C}^{\hat{t}-1}\mathcal{F}} & B_{\mathcal{C}^{\hat{t}-1}\bar{\mathcal{A}}^{\hat{t}}} & B_{\mathcal{C}^{\hat{t}-1}\mathcal{C}^{\hat{t}-1}} \end{bmatrix}}_{=:\hat{B}^{\hat{t}-1}} \underbrace{\begin{pmatrix} p_{\mathcal{F}}^{\hat{t}-1} \\ p_{\bar{\mathcal{A}}^{\hat{t}}}^{\hat{t}-1} \\ p_{\mathcal{C}^{\hat{t}-1}}^{\hat{t}-1} \end{pmatrix}}_{=:\hat{p}^{\hat{t}-1}} = - \underbrace{\begin{pmatrix} g_{\mathcal{F}} \\ g_{\bar{\mathcal{A}}^{\hat{t}}} \\ g_{\mathcal{C}^{\hat{t}-1}} \end{pmatrix}}_{=:\hat{g}^{\hat{t}-1}}.$$

With (3.17) we obtain

$$(g_{\mathcal{C}^{\hat{t}-1}})^T p^{\hat{t}-1}_{\mathcal{C}^{\hat{t}-1}} = (g^{\hat{t}-1})^T \hat{p}^{\hat{t}-1} = -(\hat{p}^{\hat{t}-1})^T B^{\hat{t}-1} \hat{p}^{\hat{t}-1} < 0 \tag{3.18}$$

because B is positive definite and $p^{\hat{t}-1} \neq 0$.

On the other hand, $\mathcal{C}^{\hat{t}-1} \subseteq \bar{\mathcal{F}} \setminus \mathcal{A}^{\hat{t}-1} \subseteq \bar{\mathcal{F}} \setminus \mathcal{A}^{\hat{t}-2} \subseteq \ldots \subseteq \bar{\mathcal{F}} \setminus \mathcal{A}^0 = \bar{\mathcal{F}} \setminus \mathcal{B}(g)$. From (3.4) we then have $g_i < 0$ for all $i \in \mathcal{C}^{\hat{t}-1}$. Also, from the definition of $\mathcal{C}^{\hat{t}-1}$, it is $p_i^{\hat{t}-1} < 0$ for all $i \in \mathcal{C}^{\hat{t}-1}$. Therefore,

$$(g_{\mathcal{C}^{\hat{t}-1}})^T p^{\hat{t}-1}_{\mathcal{C}^{\hat{t}-1}} = \sum_{i \in \mathcal{C}^{\hat{t}-1}} g_i p_i^{\hat{t}-1} > 0,$$

in contradiction to (3.18).

It follows that $\hat{t}$ must be zero, and (3.17) yields that $g_i = 0$ for any $i \notin \mathcal{A}^0 = \mathcal{B}(\nabla f(x^k))$. Consequently, (3.5) holds. □

3.2.5. Line search

Once a search direction p^k at an iterate x^k has been calculated, the algorithm determines a step size $\alpha^k > 0$ to generate the next iterate, $x^{k+1} = P(x^k + \alpha^k p^k)$. The projection ensures that the new iterate is feasible.

For the unconstrained minimization of a nonsmooth function, Lewis and Overton (Lewis and Overton, 2013) use the weak Wolfe conditions (3.19) and (3.20) to determine whether a trial point $x^{\text{trial}} = x^k + \alpha p^k$ is acceptable as a new iterate. Given fixed values for $c_1, c_2 \in (0, 1)$ with $c_1 < c_2$, the first condition,

$$f(x^{\text{trial}}) \leq f(x^k) + \alpha c_1 \nabla f(x^k)^T p^k, \tag{3.19}$$

ensures that the objective function decreases by at least a fraction of what is predicted by a linear approximation that is based on the gradient. Because the objective is nonsmooth, the linear model might be a good approximation only for very small step sizes α. Nevertheless, since f is assumed to be differentiable at x^k, condition (3.19) can be satisfied as long as α is sufficiently small.

The second condition,

$$\nabla f(x^{\text{trial}})^T p^k \geq c_2 \nabla f(x^k)^T p^k, \tag{3.20}$$

imposes that the slope of the function $\phi(\alpha) = f(x^k + \alpha p^k)$ is less steep at α than at 0, indicating that sufficient progress towards a local minimizer of $\phi(\alpha)$ is made. With a nonsmooth objective, a local minimizer of $\phi(\alpha)$ might be at a point where f (and hence ϕ) is nondifferentiable. If the current iterate is close to such a point, requiring (3.20) leads to a step size α that is beyond the point of nondifferentiability. This observation is crucial and provides the main intuition why the BFGS algorithm works well for nonsmooth problems. Because the next iterate lies on another "smooth piece" of the nonsmooth function, the new gradient is quite different from the current gradient, even when the next iterate is very nearby. Recalling the definition (3.11) of s^k and y^k in the BFGS update, we see that then the gradient difference y^k is much larger in size than the step s^k. Consequently, high curvature in the direction s^k is incorporated into the BFGS update B^{k+1}, approximating the infinite curvature at the point of nondifferentiability.

Lewis and Overton (Lewis and Overton, 2013) prove that, in the absence of bounds and under smoothness assumptions, a step size α satisfying both (3.19) and (3.20) always exists, and they provide a bracketing procedure to find it. We point out that (3.20) also

guarantees that the (s^k, y^k) pair for the BFGS update satisfies $(s^k)^T y^k > 0$ so that the update is well-defined.

On the other hand, for the minimization of a *smooth* objective function subject to bound constraints, Ferry (Ferry, 2011) proposed a generalized Wolfe line search which replaces the search direction by $\bar{p}^k = T(x^k, p^k)$. Recalling the definition of T in (3.3), we see that $\bar{p}^k$ is the modified search direction that zeros out all components that would result in an immediate violation of a constraint. With this, the Wolfe conditions suggested by Ferry (Ferry, 2011) are

$$f(x^{\text{trial}}) \leq f(x^k) + \alpha c_1 \nabla f(x^k)^T \bar{p}^k \tag{3.21}$$

$$\nabla f(x^{\text{trial}})^T T(x^{\text{trial}}, p^k) \geq c_2 \nabla f(x^k)^T \bar{p}^k. \tag{3.22}$$

We adopt these conditions in our context of minimizing a nonsmooth objective. Algorithm 4 describes the corresponding bracketing mechanism.

Note that, in the absence of bounds, this procedure is identical to the line search algorithm proposed by Lewis and Overton (Lewis and Overton, 2013). When f is smooth, Ferry (Ferry, 2011) showed that there always exists a step size α that satisfies both (3.21) and (3.22). For a nonsmooth objective, such a step size may not exist. In our method, when a suitable step size cannot be found in step 16, finite termination is ensured by the termination test in step 24.

The bracketing mechanism generates a sequence of values for U and L in a way that shrinks the length of the interval $[L, U]$ to zero (see steps 20 and 22 together with steps 11 and 14). Because it is not clear whether a step size satisfying both (3.21) and (3.22) can be found, the algorithm will attempt only a moderate number of trial step sizes, until the

Algorithm 4 ModifiedWolfe

Inputs: Current iterate x and a search direction p.

Output: Step size α to generate next iterate.

Parameters: $c_1, c_2 \in (0,1)$ with $c_1 < c_2$; $\epsilon_{\text{abs}}, \epsilon_{\text{rel}} > 0$.

1: Set $L = 0$

2: Set $U = \max_i\{\gamma_i\}$, where $\gamma_i \stackrel{\text{def}}{=} \begin{cases} \frac{u_i - x_i}{p_i} & p_i > 0 \text{ and } x_i \neq u_i; \\ \frac{x_i - l_i}{p_i} & p_i < 0 \text{ and } x_i \neq l_i; \\ \infty & \text{otherwise.} \end{cases}$

3: Set $\alpha = \min(1, U)$.

4: Compute $\bar{p} = T(x, p)$.

5: **if** $\bar{p} = 0$ **then**

6: **terminate** with "No search direction". ▷ Error due to bad search direction

7: **end if**

8: **for** $t = 0, 1, 2, \ldots$ **do**

9: Set $x^{\text{trial}} = P(x + \alpha\bar{p})$.

10: **if** (3.21) does not hold **then** ▷ Sufficient decrease condition does not hold

11: Set $U = \alpha$.

12: **else**

13: **if** (3.22) does not hold **then** ▷ Curvature condition does not hold

14: Set $L = \alpha$.

15: **else**

16: **return** α. ▷ Return step satisfying weak Wolfe conditions

17: **end if**

18: **end if**

relative interval length is on the order of ϵ_{rel}. Whenever a step size is encountered that satisfies the sufficient decrease condition (3.21), step 14 sets L to this value (unless the search is terminated in step 16). Therefore, when step 26 returns a nonzero step size, it is

Algorithm 4 ModifiedWolfe

19: **if** $U < \max_i\{\gamma_i\}$ **then** ▷ Update step-size
20: Set $\alpha = \frac{U+L}{2}$.
21: **else**
22: Set $\alpha = \min(2L, U)$.
23: **end if**
24: **if** $U - L < \epsilon_{\text{abs}} + \epsilon_{\text{rel}} L$ **then**
25: **if** $L > 0$ **then**
26: **return** L. ▷ Return step satisfying sufficient decrease condition
27: **else**
28: **terminate** with "Line Search Error". ▷ Error, no suitable step size found
29: **end if**
30: **end if**
31: **end for**

guaranteed that the next iterate will have a smaller objective value, and so the overall optimization algorithm cannot cycle. On the other hand, if L is zero in step 28, the trial step size $U = \alpha$ has become smaller than ϵ_{abs}. In that case, we declare a line search error, which is likely caused by numerical issues in the search direction computation or round-off in the function evaluation. Finally, it may happen that the computed search direction is such that $P(x + \alpha p) = x$ for any $\alpha > 0$. Then there is no point in conducting a line search and we terminate the optimization with an error message in step 6. This may occur due to numerical problems during the step computation.

Algorithm 5 Nonsmooth Quasi-Newton (NQN)

Inputs: Initial point $x^0 \in [l, u]$.

Parameters: Size of L-BFGS memory m; update tolerance ϵ_{skip}.

1: Initialize storage $\mathcal{S}$ of L-BFGS curvature pairs to be empty.
2: **for** $k = 0, 1, 2, \ldots$ **do**
3: **if** $T(x^k, -\nabla f(x^k) = 0$ **then**
4: **return** x^k ▷ Finite termination at stationary point
5: **end if**
6: Choose active set $\mathcal{A}^k$. ▷ Details specified elsewhere
7: Compute search direction p^k.
8: Compute $\alpha^k = \texttt{ModifiedWolfe}(x^k, p^k)$. ▷ Perform line search using Algorithm 2
9: Set $\bar{p}^k = T(x^k, p^k)$.
10: Set $x^{k+1} = P(x^k + \alpha^k \bar{p}^k)$.
11: Compute curvature pair (s^k, y^k) from (3.11).
12: **if** $(s^k)^T y^k > \epsilon_{\text{skip}} \|s^k\| \|y^k\|$ **then** ▷ Discard pair if curvature condition not satisfied
13: Store (s^k, y^k) in $\mathcal{S}$. ▷ Update L-BFGS memory
14: If $|\mathcal{S}| > m$ then discard oldest curvature pair.
15: **end if**
16: **end for**

3.2.6. Main Algorithm

The overall optimization algorithm for solving (3.1) is given in Algorithm 5. Step 6 is purposely left vague, because we will explore different alternatives for the active-set selection. The experiments in the following section consider the following options:

Variant 1: Choose the active set based on the gradient at the current iterate, $\mathcal{A}^k = \mathcal{B}(\nabla f(x^k))$.

Variant 2: Choose the active set based on the subgradient approximation using (3.9).

Variant 3: Compute the active set from the correction procedure Algorithm 3 with initial guess $\mathcal{A}^{\text{init}} = \mathcal{B}(\nabla f(x^k))$.

Variant 4: Compute the active set from the correction procedure Algorithm 3 with initial guess $\mathcal{A}^{\text{init}}$ based on the subgradient approximation using (3.9).

For the last two variants, the search direction is already computed as byproduct of the active set selection and step 7 does not require any actual work.

There is no guarantee that the pair (s^k, y^k) pair defined in (3.11) satisfies the curvature condition (3.12), even when the weak Wolfe conditions (3.21) and (3.22) are satisfied, since the actual step s^k might be different from $\alpha^k \bar{p}^k$, due to the projection in step 10. This is in contrast to the unconstrained case, where (3.22) implies that (3.11) holds. To handle this situation, the update is skipped in step 12 whenever $(s^k)^T(y^k) \leq \epsilon_{\text{skip}} \|s^k\| \|y^k\|$.

As mentioned in Section 1, we do not consider any theoretical convergence properties of this method, including the possibilities of stalling at non-stationary points and of spurious termination of the line search. We point out that convergence guarantees remain an open question even when no constraints are present.

3.3. Numerical Experiments

3.3.1. Implementation and Problem Set

We implemented Algorithm 5 in Python. We will refer to it as NQN. The code for our algorithm can be found in our GitHub repository: `https://github.com/keskarnitish/NQN`. The values for the various parameters used are summarized in Table 3.1. In order to solve the quadratic program (3.8) for the subgradient approximation, we used the

Parameter	Value	Description
m	20	L-BFGS memory
M	20	Maximum size of sample set G^k
(c_1, c_2)	$(10^{-8}, 0.9)$	Parameters on Wolfe conditions
ϵ_{abs}	10^{-16}	Absolute bracketing tolerance
ϵ_{rel}	10^{-6}	Relative bracketing tolerance
ϵ_{skip}	10^{-8}	Tolerance for skipping L-BFGS update

Table 3.1. Parameter values used for numerical experiments.

CVXOPT package (Andersen et al., 2013). We rely on the NumPy package (Walt et al., 2011) for linear algebra operations, and Theano (Team et al., 2016) is used to compute derivatives of the objective functions using algorithmic differentiation.

We compare NQN with other codes for solving (3.1), namely (i) L-BFGS-B (Byrd et al., 1995); (ii) L-BFGS-B-NS (Henao, 2014); and (iii) LMBM-B (Karmitsa and Mäkelä, 2010b). While L-BFGS-B is not designed to solve nonsmooth problems, we nonetheless include it owing to its documented success at solving smooth bound-constrained problems. We include L-BFGS-B-NS and LMBM-B since they are specifically designed to solve (3.1) and have shown competitive performance on variety of tasks. The former is identical to the L-BFGS-B algorithm except that it uses the weak Wolfe line search as opposed to the strong Wolfe line search. LMBM-B (Karmitsa and Mäkelä, 2010b) combines the limited-memory bundle method (LMBM) (Haarala et al., 2004, 2007) with a Cauchy-point-based active-set selection strategy similar to the one in L-BFGS-B. The LMBM-B method generates steps using a subgradient bundle in conjunction with an L-BFGS/SR-1 updating scheme to gain curvature information.

To make sure each solver obtains the same function and derivative information, we implemented Python wrappers around the Fortran codes written by the respective authors.

The original Fortran codes can be found at `users.iems.northwestern.edu/~nocedal/lbfgsb.html`, `github.com/wilmerhenao/L-BFGS-B-NS` and `napsu.karmitsa.fi/lmbm/` for L-BFGS-B, L-BFGS-B-NS and LMBM-B respectively. We exclude other methods, including gradient-sampling methods, since we found their performance to be inferior to the methods listed above.

To explore the effect of the different active-set identification mechanisms, we propose two generalizations of (3.6) as test problems for nonsmooth optimization. The two problems are defined for even values of n; we call them `Myopic_Decoupled` and `Myopic_Coupled`. They are given as

$$\min_{x\in\mathbb{R}^n} \sum_{i\in\{1,3,\cdots,n-1\}} |x_i - x_{i+1}| + (x_i + 0.1x_{i+1})^2, \tag{3.23}$$

and

$$\min_{x\in\mathbb{R}^n} \sum_{i=1}^{n-1} |x_i - x_{i+1}| + (x_i + 0.1x_{i+1})^2, \tag{3.24}$$

respectively; the bound constraints for these problems are discussed below. The attributes `Decoupled` and `Coupled` refer to whether or not the problem is separable.

In addition, we use several test problems from the literature that are listed in Table 3.2 along with their references[1]. Since these test problems are for unconstrained optimization, we follow an approach similar to (Karmitsa and Mäkelä, 2010a) in that we add the following

[1]The objective function for problem 20, suggested by Michael Overton (Overton, 02/23/2016), is $\max\left\{|x_1|, \max_{i\in\{2,3,\cdots,n\}} |x_{i-1} - x_i|\right\}$.

Problem Number	Test Problem	Reference
1	`Active_Faces`	(Haarala et al., 2004)
2	`Chained_CB3_1`	(Haarala et al., 2004)
3	`Chained_CB3_2`	(Haarala et al., 2004)
4	`Chained_Crescent_1`	(Haarala et al., 2004)
5	`Chained_Crescent_2`	(Haarala et al., 2004)
6	`Chained_LQ`	(Haarala et al., 2004)
7	`Chained_Mifflin_2`	(Haarala et al., 2004)
8	`Convex_Nonsmooth`	(Skajaa, 2010)
9	`L1`	(Skajaa, 2010)
10	`L1HILB`	(Haarala et al., 2004)
11	`L2`	(Lewis and Overton, 2013)
12	`MAXHILB`	(Haarala et al., 2004)
13	`MAXQ`	(Haarala et al., 2004)
14	`Modified_Rosenbrock_1`	(Henao, 2014)
15	`Modified_Rosenbrock_2`	(Henao, 2014)
16	`Myopic_Coupled`	(3.24)
17	`Myopic_Decoupled`	(3.23)
18	`Nesterov_1`	(Gürbüzbalaban and Overton, 2012)
19	`Nesterov_2`	(Gürbüzbalaban and Overton, 2012)
20	`Nesterov_3`	(Overton, 02/23/2016)
21	`Nonsmooth_Brown`	(Haarala et al., 2004)
22	`TEST29_2`	(Lukšan et al., 2014)
23	`TEST29_6`	(Lukšan et al., 2014)
24	`TEST29_22`	(Lukšan et al., 2014)
25	`TEST29_24`	(Lukšan et al., 2014)

Table 3.2. Test problems used in numerical experiments.

bounds to all problems:

$$l_i = \begin{cases} [x^\star_{\text{uncon}}]_i - 5.5 & \text{if} \quad \text{mod } (i,2) = 0 \\ -100 & \text{if} \quad \text{mod } (i,2) = 1 \end{cases}$$

$$u_i = \begin{cases} [x^\star_{\text{uncon}}]_i - 0.5 & \text{if} \quad \text{mod } (i,2) = 0 \\ 100 & \text{if} \quad \text{mod } (i,2) = 1 \end{cases}$$

where $x^\star_{\text{uncon}}$ is the unconstrained global minimizer which is known for all problems in closed form. By construction, for all problems, the unconstrained minimizer lies outside the bounds. For each of the 25 problems, ten starting points were generated randomly via a uniform distribution $U(-2, 2)$ centered at the midpoint of the bounds, giving rise to a total of 250 instances. Each code was run until the number of gradient evaluations exceeded $100n$ or an error occurred.

3.3.2. Effect of Active-Set Prediction and Correction Mechanism

We begin by investigating the efficacy of the active-set prediction, the correction mechanism, and their interplay, using the four variants given in Section 3.2.6.

Let us first consider the `Myopic_Decoupled` and `Myopic_Coupled` problems with $n = 100$ for one particular starting point. These problems are designed specifically to highlight the failure of gradient-based active-set prediction strategies. Figures 3.2 and 3.3 detail the behavior of the algorithm over the course of the optimization. In each plot, the dotted blue line depicts the progress in objective function, measured as $(f(x^k) - f(x^\star))/(f(x^0) - f(x^\star))$, where $x^\star$ is the optimal solution. The solid red line gives the size of the (initial) active set $\mathcal{A}^k$. For the variants that employ the active set correction strategy, the dashed red line gives the size of the active set at the end of Algorithm 3.

As one might expect, Variant 1, which uses the myopic gradient and no correction strategy, fails to identify the optimal active set (which contains 50 variables) and its guess $\mathcal{A}^k$ keeps fluctuating. The reduction in the objective function is significantly slower compared to the other variants. When the subgradient approximation is used in Variant 2, the objective function decreases faster, and after a certain number of iterations, the active

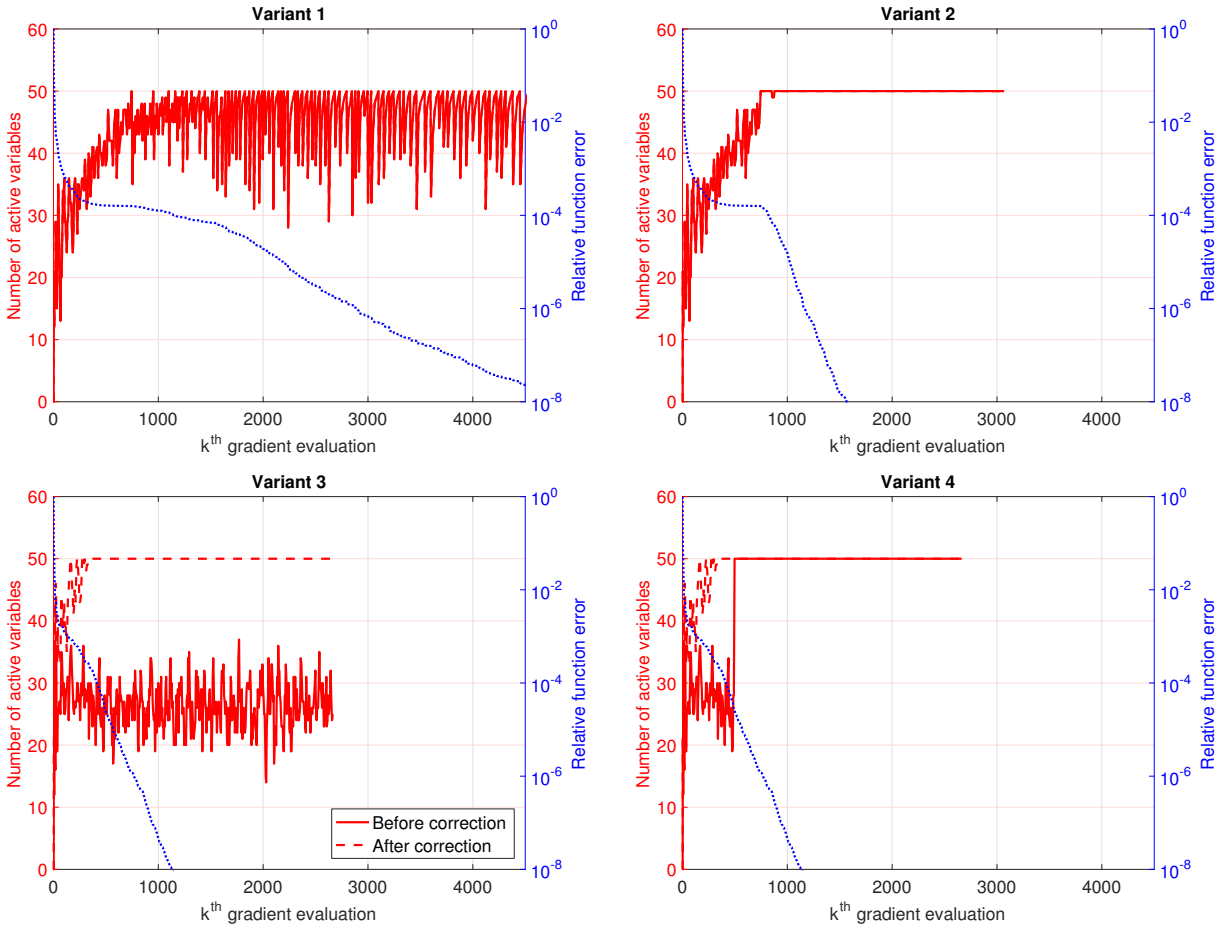


Figure 3.2. A comparison of the four variants of the algorithm on the `Myopic_Decoupled` problem.

set settles to the optimal active set. When the correction strategy is used in Variants 3 and 4, the optimal active set is identified more quickly, and the reduction in the objective is even faster. In these experiments, there is very little difference in performance between the two initializations of $\mathcal{A}^{\text{init}}$ in Algorithm 3. We observe similar behavior for larger dimensions of this problem. Further, rapid fluctuations in the active set are also seen in the other algorithms (L-BFGS-B, L-BFGS-B-NS and LMBM-B) which also employ gradient-based identification strategies.

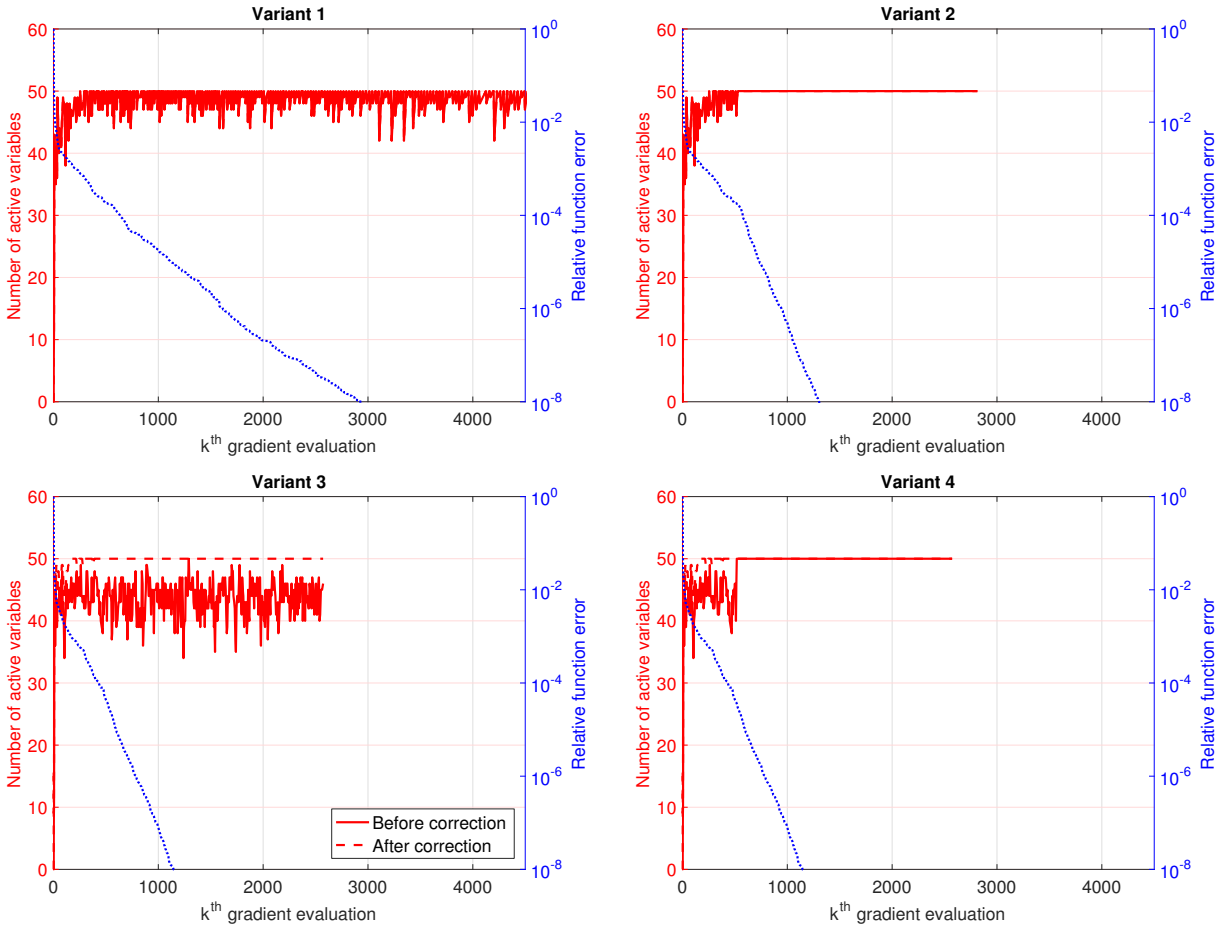


Figure 3.3. A comparison of the four variants of the algorithm on the `Myopic_Coupled` problem.

Next we assess the relative performance of the different variants for the entire set of 250 instances with $n = 100$. Figure 3.4 presents Dolan-Moré performance profiles (Dolan and Moré, 2002) with respect to the number of gradient evaluations. These profiles rely on a condition to determine when a run is deemed converged. For this purpose, given a tolerance $\epsilon > 0$, we use the test

$$\frac{f(x^k) - f^\star}{f(x^0) - f^\star} < \epsilon \tag{3.25}$$

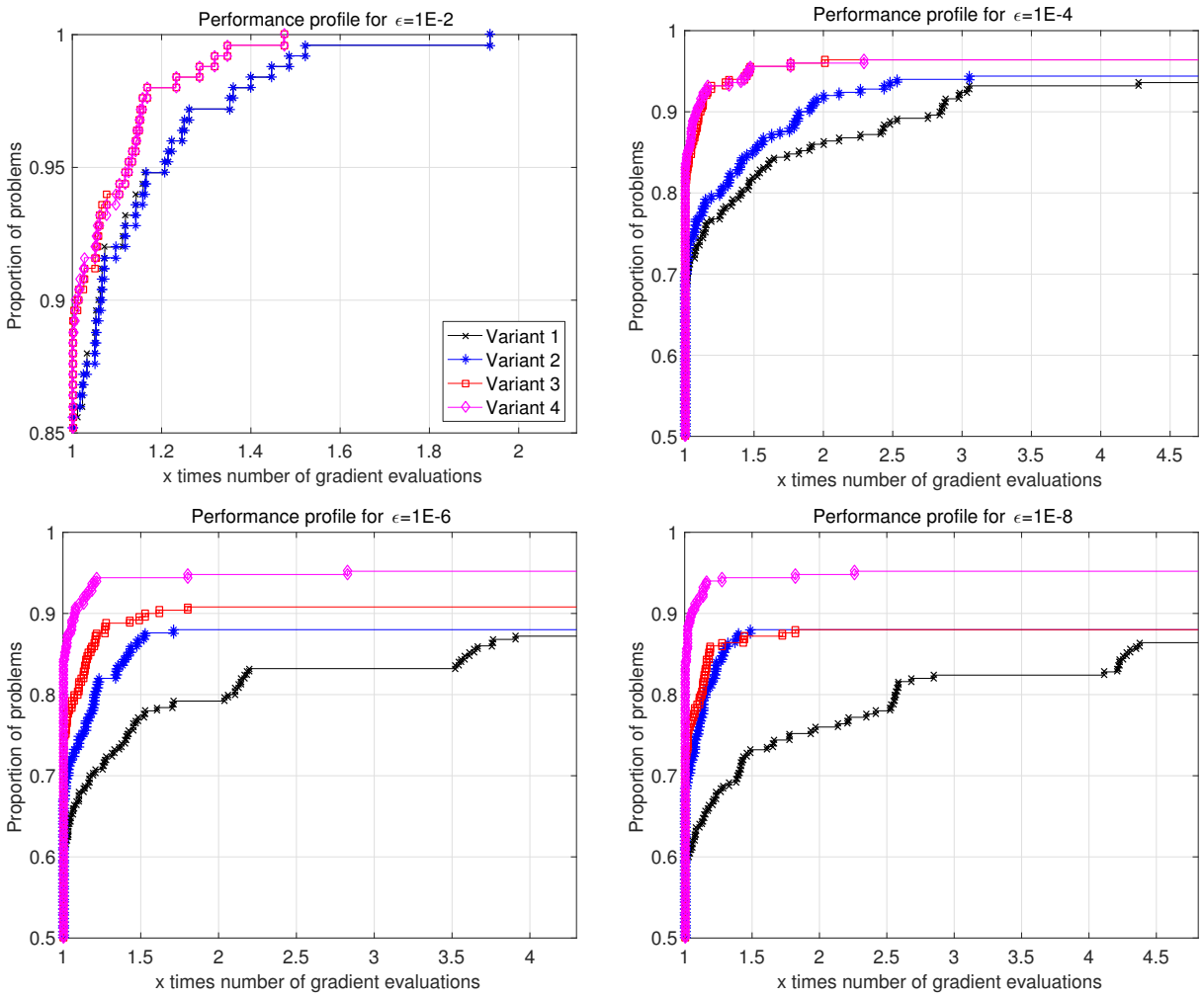


Figure 3.4. Dolan-Moré performance profiles comparing the four variants of the algorithm on 250 test problems for $\epsilon = 10^{-2}, 10^{-4}, 10^{-6}$ and $\epsilon = 10^{-8}$.

where $f^\star$ is the best value found by any of the methods for the same instance. We present plots for four values of ϵ viz., $10^{-2}, 10^{-4}$, 10^{-6} and 10^{-8}.

This experiment reveals results similar to those found in Figures 3.2 and 3.3. Variant 1 shows the worst behavior, and the use of the subgradient approximation in Variant 2 improves the convergence rate. Using the corrective strategy gives the best performance, with an advantage for Variant 4 when a very tight tolerance is used.

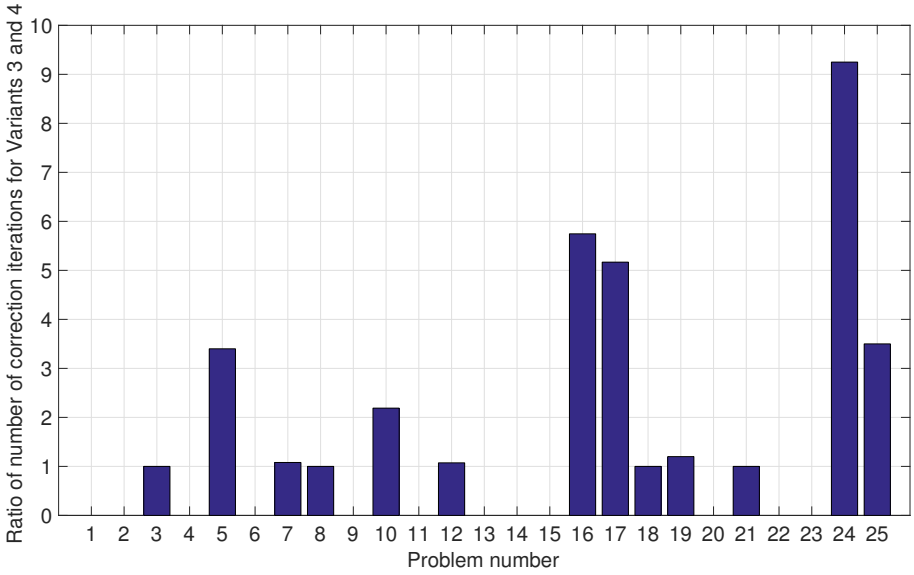


Figure 3.5. Average ratio of number of corrections for Variants 3 and 4 for all problems for $\epsilon = 10^{-4}$ and $n = 100$. A ratio of 0 indicates that both methods did not need any corrections.

Variants 3 and 4 incur different computational costs per iteration in Algorithm 5. In addition to the cost of the corrective loop, Variant 4 relies on the solution of the quadratic program (3.9) for the computation of the subgradient approximation. Figures 3.2 and 3.3 suggest that Variant 4 might require fewer iterations in the corrective loop in Algorithm 3 than Variant 3, since its initial active set $\mathcal{A}^{\text{init}}$ is a better guess of the final active set returned by the correction procedure. In Figure 3.5, we present the average ratio of the number of correction iterations for Variants 3 over 4. Indeed, Variant 3 needs up to 10 times as many correction iterations as Variant 4 to achieve similar performance.

Nevertheless, since the solution of the quadratic program (3.9) comes at a significant computational cost, we used Variant 3 for the remaining experiments. Also, Variant 3 is consistent with Lemma 3.2.1, so that we would encounter a zero step from the correction loop only when the current iterate is already stationary.

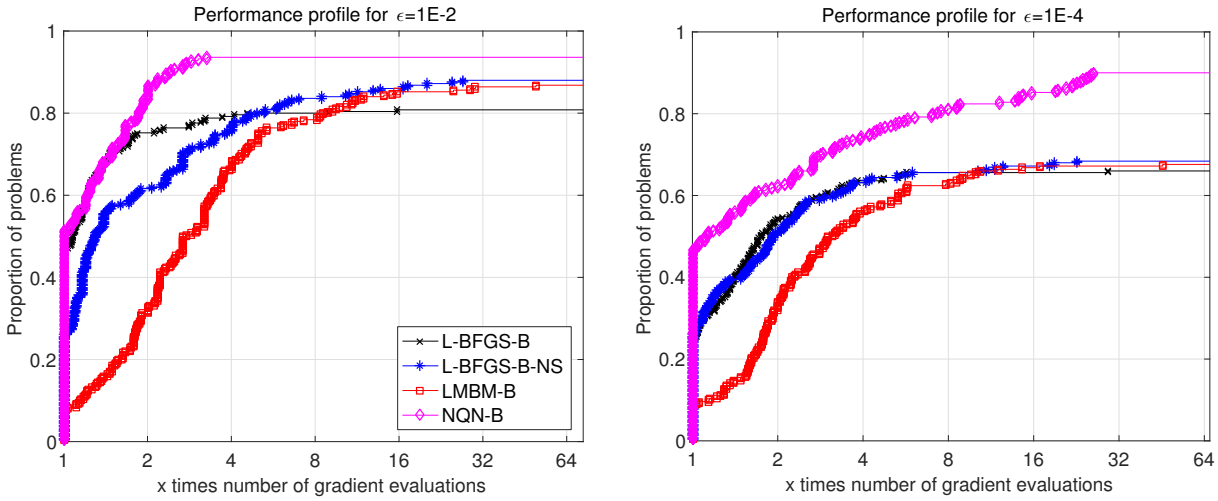


Figure 3.6. Dolan-Moré performance profiles of gradient evaluations for 250 test problems for $\epsilon = 10^{-2}$ and $\epsilon = 10^{-4}$ with $n = 100$.

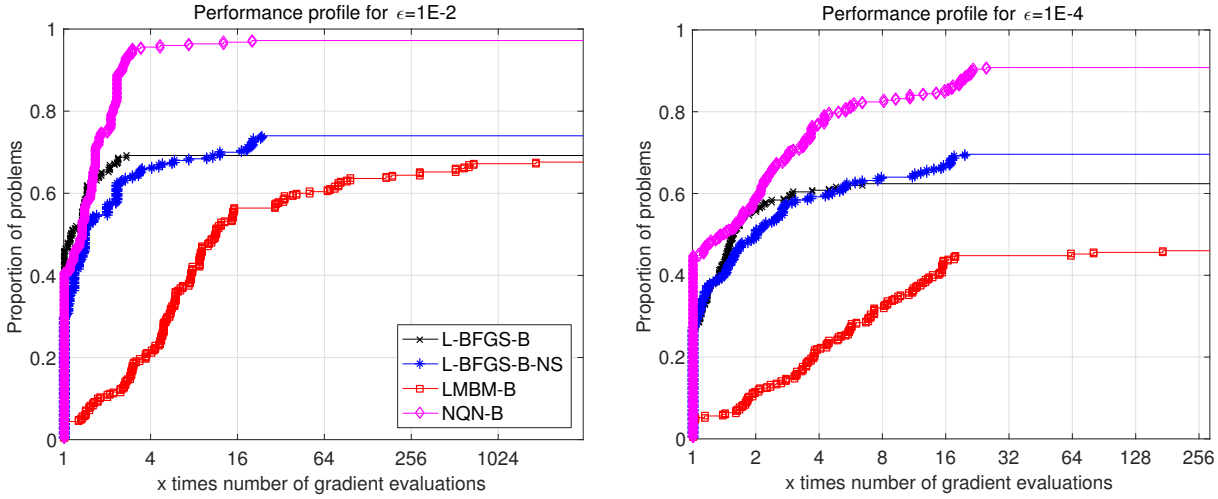


Figure 3.7. Dolan-Moré performance profiles of gradient evaluations for 250 test problems for $\epsilon = 10^{-2}$ and $\epsilon = 10^{-4}$ with $n = 1000$.

3.3.3. Comparison with Other Methods

We now compare NQN with L-BFGS-B, L-BFGS-B-NS, and LMBM-B on the 250 test instances. Figures 3.6, 3.7, and 3.8 correspond to three sets of experiments, with $n = 100$, $n = 1000$, and $n = 10000$, respectively. We present performance profiles for two values

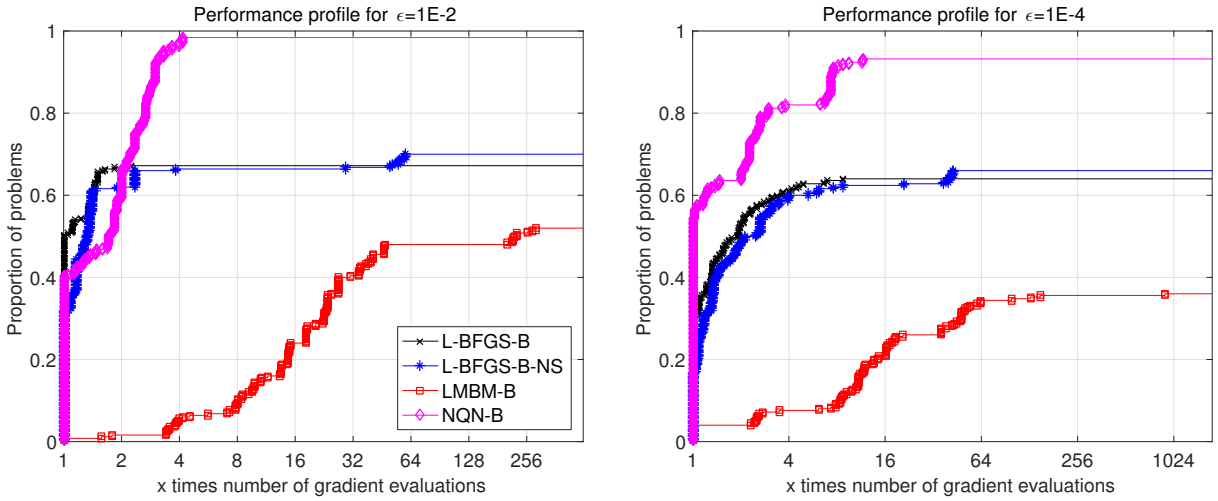


Figure 3.8. Dolan-Moré performance profiles of gradient evaluations for 250 test problems for $\epsilon = 10^{-2}$ and $\epsilon = 10^{-4}$ with $n = 10000$.

of ϵ, viz. 10^{-2} and 10^{-4}. We do not report experiments for $\epsilon = 10^{-6}$ and $\epsilon = 10^{-8}$ since the relative error is based on the best function value obtained by any of the methods. Tighter tolerances for ϵ would magnify insignificant differences between methods when neither are very close to an optimal solution. It can be seen that the proposed algorithm performs better than the other methods across different tolerances ϵ and problem sizes n. The figures show that NQN is able to find a lower objective on more problems and requires fewer gradient evaluations. This difference is particularly pronounced for tight tolerances and large problem sizes.

In Table 3.3, we summarize the occurrence for failures of the various methods for tolerances of $\epsilon = 10^{-2}$ and $\epsilon = 10^{-4}$. The flag `OK` indicates that the termination criterion was satisfied at some iteration, `MAX` corresponds to reaching maximum number of gradient evaluations, and `OTHER` implies other failures which include solver-specific causes such as spurious termination of the line search, numerical issues, or convergence to a non-stationary point. For NQN we break down the number of `OTHER` failures into convergence to a point

Flag	OK	MAX	OTHER	OK	MAX	OTHER
	$\epsilon = 10^{-2}$			$\epsilon = 10^{-4}$		
$n = 100$						
L-BFGS-B	202	0	48	165	0	85
L-BFGS-B-NS	220	25	5	171	58	21
LMBM-B	217	16	17	169	64	17
NQN	234	12	4 + 0	225	17	7 + 1
$n = 1000$						
L-BFGS-B	173	4	73	156	8	86
L-BFGS-B-NS	185	61	4	174	68	8
LMBM-B	169	11	70	115	54	81
NQN	243	6	1 + 0	227	22	1 + 0
$n = 10000$						
L-BFGS-B	168	17	65	160	21	69
L-BFGS-B-NS	175	73	2	165	82	3
LMBM-B	130	30	90	90	70	90
NQN	246	2	2 + 0	233	15	2 + 0

Table 3.3. Number of outcomes with different termination messages.

with no feasible direction in step 6 of Algorithm 4 (first number) and line search failure in step 28 of Algorithm 4 (second number).

As can be seen from Table 3.3, failures for NQN are more often due to budget exhaustion rather than another type of failure. In total, there were 10 instances in which numerical issues led to a bad search direction. A line search error was observed only once. The cause for budget exhaustion in NQN is, in part, due to the tight tolerance of ϵ_{abs}; close to a solution, the bracketing procedure takes many iterations in order to find points providing sufficient function decrease. Most of the large number of failures for L-BFGS-B occur due to a breakdown in the line search. This is to be expected since L-BFGS-B employs a strong Wolfe line search which is difficult to be satisfied with a nonsmooth objective. When the weak Wolfe line search is used in L-BFGS-B-NS instead, the number of line search failures is reduced significantly. Nevertheless, the overall number of successfully

solved problems increases only marginally. LMBM-B is the least robust method, with a noticeable increase in the failure rate as the problem size grows.

CHAPTER 4

On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima

4.1. Introduction

Deep Learning has emerged as one of the cornerstones of large-scale machine learning. Deep Learning models are used for achieving state-of-the-art results on a wide variety of tasks including computer vision, natural language processing and reinforcement learning; see (Bengio et al., 2016) and the references therein. The problem of training these networks is one of non-convex optimization. Mathematically, this can be represented as:

$$\min_{x \in \mathbb{R}^n} \quad f(x) := \frac{1}{M} \sum_{i=1}^{M} f_i(x), \tag{4.1}$$

where f_i is a loss function for data point $i \in \{1, 2, \cdots, M\}$ which captures the deviation of the model prediction from the data, and x is the vector of weights being optimized. The process of optimizing this function is also called *training* of the network. Stochastic Gradient Descent (SGD) (Bottou, 1998; Sutskever et al., 2013) and its variants are often used for training deep networks. These methods minimize the objective function f by iteratively taking steps of the form:

$$x_{k+1} = x_k - \alpha_k \left(\frac{1}{|B_k|} \sum_{i \in B_k} \nabla f_i(x_k) \right), \tag{4.2}$$

where $B_k \subset \{1, 2, \cdots, M\}$ is the batch sampled from the data set and α_k is the step size at iteration k. These methods can be interpreted as gradient descent using noisy gradients, which and are often referred to as mini-batch gradients with batch size $|B_k|$. SGD and its variants are employed in a small-batch regime, where $|B_k| \ll M$ and typically $|B_k| \in \{32, 64, \cdots, 512\}$. These configurations have been successfully used in practice for a large number of applications; see e.g. (Simonyan and Zisserman, 2014; Graves et al., 2013; Mnih et al., 2013). Many theoretical properties of these methods are known. These include guarantees of: (a) convergence to minimizers of strongly-convex functions and to stationary points for non-convex functions (Bottou et al., 2016), (b) saddle-point avoidance (Ge et al., 2015; Lee et al., 2016), and (c) robustness to input data (Hardt et al., 2015).

Stochastic gradient methods have, however, a major drawback: owing to the sequential nature of the iteration and small batch sizes, there is limited avenue for parallelization. While some efforts have been made to parallelize SGD for Deep Learning (Dean et al., 2012; Das et al., 2016; Zhang et al., 2015), the speed-ups and scalability obtained are often limited by the small batch sizes. One natural avenue for improving parallelism is to increase the batch size $|B_k|$. This increases the amount of computation per iteration, which can be effectively distributed. However, practitioners have observed that this leads to a loss in generalization performance; see e.g. (LeCun et al., 2012). In other words, the performance of the model on testing data sets is often worse when trained with large-batch methods as compared to small-batch methods. In our experiments, we have found the drop in generalization (also called generalization gap) to be as high as 5% even for smaller networks.

In this paper, we present numerical results that shed light into this drawback of large-batch methods. We observe that the generalization gap is correlated with a marked sharpness of the minimizers obtained by large-batch methods. This motivates efforts at remedying the generalization problem, as a training algorithm that employs large batches without sacrificing generalization performance would have the ability to scale to a much larger number of nodes than is possible today. This could potentially reduce the training time by orders-of-magnitude; we present an idealized performance model in the Appendix A.2 to support this claim.

The paper is organized as follows. In the remainder of this section, we define the notation used in this paper, and in Section 4.2 we present our main findings and their supporting numerical evidence. In Section 4.3 we explore the performance of small-batch methods, and in Section 4.5 we briefly discuss the relationship between our results and recent theoretical work. We conclude with open questions concerning the generalization gap, sharp minima, and possible modifications to make large-batch training viable. In Section 4.4, we present some attempts to overcome the problems of large-batch training.

4.1.1. Notation

We use the notation f_i to denote the composition of loss function and a prediction function corresponding to the i^{th} data point. The vector of weights is denoted by x and is subscripted by k to denote an iteration. We use the term small-batch (SB) method to denote SGD, or one of its variants like ADAM (Kingma and Ba, 2015) and ADAGRAD (Duchi et al., 2011), with the proviso that the gradient approximation is based on a small mini-batch. In our setup, the batch B_k is randomly sampled and its size is kept fixed for every iteration.

We use the term large-batch (LB) method to denote any training algorithm that uses a large mini-batch. In our experiments, ADAM is used to explore the behavior of both a small or a large batch method.

4.2. Drawbacks of Large-Batch Methods

4.2.1. Our Main Observation

As mentioned in Section 4.1, practitioners have observed a generalization gap when using large-batch methods for training deep learning models. Interestingly, this is despite the fact that large-batch methods usually yield a similar value of the training function as small-batch methods. One may put forth the following as possible causes for this phenomenon: (i) LB methods over-fit the model; (ii) LB methods are attracted to saddle points; (iii) LB methods lack the *explorative* properties of SB methods and tend to zoom-in on the minimizer closest to the initial point; (iv) SB and LB methods converge to qualitatively different minimizers with differing generalization properties. The data presented in this paper supports the last two conjectures.

The main observation of this paper is as follows:

> The lack of generalization ability is due to the fact that large-batch methods tend to converge to *sharp minimizers* of the training function. These minimizers are characterized by a significant number of large positive eigenvalues in $\nabla^2 f(x)$, and tend to generalize less well. In contrast, small-batch methods converge to *flat minimizers* characterized by having numerous small eigenvalues of $\nabla^2 f(x)$ (and fewer large eigenvalues). We have observed that the loss function landscape of deep neural networks

is such that large-batch methods are attracted to regions with sharp minimizers and that, unlike small-batch methods, are unable to escape basins of attraction of these minimizers.

The concept of sharp and flat minimizers have been discussed in the statistics and machine learning literature. (Hochreiter and Schmidhuber, 1997) (informally) define a flat minimizer $\bar{x}$ as one for which the function varies slowly in a relatively large neighborhood of $\bar{x}$. In contrast, a sharp minimizer $\hat{x}$ is such that the function increases rapidly in a small neighborhood of $\hat{x}$. A flat minimum can be described with low precision, whereas a sharp minimum requires high precision. The large sensitivity of the training function at a sharp minimizer negatively impacts the ability of the trained model to generalize on new data; see Figure 4.1 for a hypothetical illustration. This can be explained through the lens of the minimum description length (MDL) theory, which states that statistical models that require fewer bits to describe (i.e., are of low complexity) generalize better (Rissanen, 1983). Since flat minimizers can be specified with lower precision than to sharp minimizers, they tend to have better generalization performance. Alternative explanations are proffered through the Bayesian view of learning (MacKay, 1992), and through the lens of free Gibbs energy; see e.g. (Chaudhari et al., 2016).

4.2.2. Numerical Experiments

In this section, we present numerical results to support the observations made above. To this end, we make use of the visualization technique employed by (Goodfellow et al., 2014b) and a proposed heuristic metric of sharpness (Equation (4.4)). We consider 6

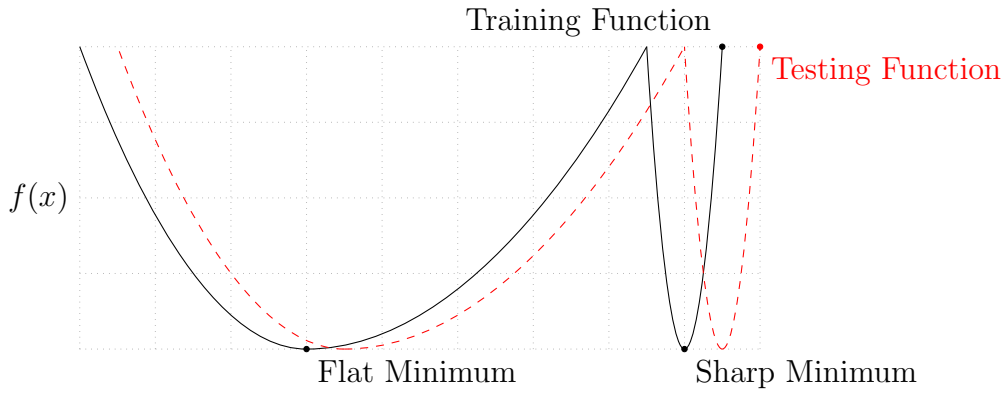


Figure 4.1. A Conceptual Sketch of Flat and Sharp Minima. The Y-axis indicates value of the loss function and the X-axis the variables (parameters)

multi-class classification network configurations for our experiments; they are described in Table 4.1. The details about the data sets are presented below in Table 4.2 and network configurations are presented in Appendix A.1. As is common for such problems, we use the mean cross entropy loss as the objective function f.

Table 4.1. Network Configurations

Name	Network Type	Architecture	Data set
F_1	Fully Connected	Section A.1.1	MNIST
F_2	Fully Connected	Section A.1.2	TIMIT
C_1	(Shallow) Convolutional	Section A.1.3	CIFAR-10
C_2	(Deep) Convolutional	Section A.1.4	CIFAR-10
C_3	(Shallow) Convolutional	Section A.1.3	CIFAR-100
C_4	(Deep) Convolutional	Section A.1.4	CIFAR-100

The TIMIT data set was pre-processed using Kaldi (Povey et al., 2011) while all others were used in their original form. The networks were chosen to exemplify popular configurations used in practice like AlexNet (Krizhevsky et al., 2012) and VGGNet (Simonyan and Zisserman, 2014). Results on other networks and using other initialization strategies, activation functions, and data sets showed similar behavior. Since the goal of

Table 4.2. Data Sets

Data Set	#Data Points Train	Test	#Features	#Classes	Reference
MNIST	60000	10000	28×28	10	(LeCun et al., 1998a,b)
TIMIT	721329	310621	360	1973	(Garofolo et al., 1993)
CIFAR-10	50000	10000	32×32	10	(Krizhevsky and Hinton, 2009)
CIFAR-100	50000	10000	32×32	100	(Krizhevsky and Hinton, 2009)

our work is not to achieve state-of-the-art accuracy or time-to-solution on these tasks but rather to characterize the *nature* of the minima for LB and SB methods, we only describe the final testing accuracy in the main paper and ignore convergence trends.

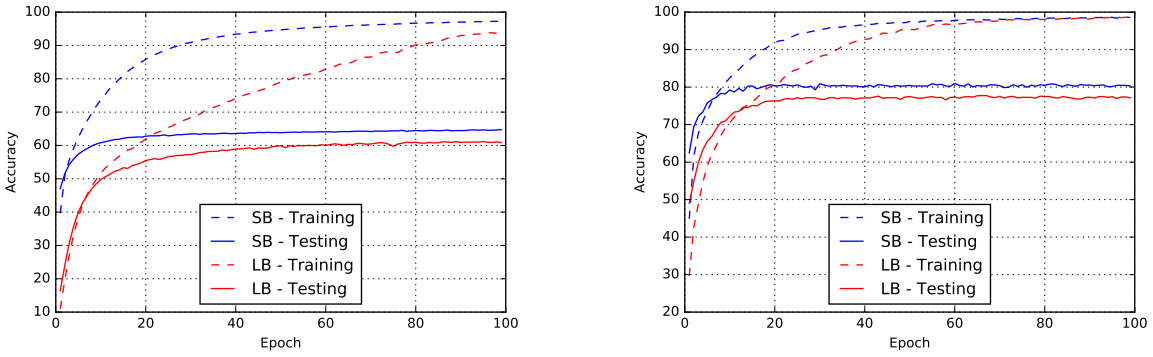
For all experiments, we used 10% of the training data as batch size for the large-batch experiments and 256 data points for small-batch experiments. We used the ADAM optimizer for both regimes. Experiments with other optimizers for the large-batch experiments, including ADAGRAD (Duchi et al., 2011), SGD (Sutskever et al., 2013) and adaQN (Keskar and Berahas, 2016), led to similar results. All experiments were conducted 5 times from different (uniformly distributed random) starting points and we report both mean and standard-deviation of measured quantities. The baseline performance for our setup is presented Table 4.3. From this, we can observe that on all networks, both approaches led to high training accuracy but there is a significant difference in the generalization performance. The networks were trained, without any budget or limits, until the loss function ceased to improve.

We emphasize that the generalization gap is not due to *over-fitting* or *over-training* as commonly observed in statistics. These phenomena manifest themselves in the form of a testing accuracy curve that, at a certain iterate peaks, and then decays due to the model learning idiosyncrasies of the training data. This is not what we observe in our

Table 4.3. Performance of small-batch (SB) and large-batch (LB) variants of ADAM on the 6 networks listed in Table 4.1

	Training Accuracy		Testing Accuracy	
Name	SB	LB	SB	LB
F_1	$99.66\% \pm 0.05\%$	$99.92\% \pm 0.01\%$	$98.03\% \pm 0.07\%$	$97.81\% \pm 0.07\%$
F_2	$99.99\% \pm 0.03\%$	$98.35\% \pm 2.08\%$	$64.02\% \pm 0.2\%$	$59.45\% \pm 1.05\%$
C_1	$99.89\% \pm 0.02\%$	$99.66\% \pm 0.2\%$	$80.04\% \pm 0.12\%$	$77.26\% \pm 0.42\%$
C_2	$99.99\% \pm 0.04\%$	$99.99\% \pm 0.01\%$	$89.24\% \pm 0.12\%$	$87.26\% \pm 0.07\%$
C_3	$99.56\% \pm 0.44\%$	$99.88\% \pm 0.30\%$	$49.58\% \pm 0.39\%$	$46.45\% \pm 0.43\%$
C_4	$99.10\% \pm 1.23\%$	$99.57\% \pm 1.84\%$	$63.08\% \pm 0.5\%$	$57.81\% \pm 0.17\%$

experiments; see Figure 4.2 for the training–testing curve of the F_2 and C_1 networks, which are representative of the rest. As such, early-stopping heuristics aimed at preventing models from over-fitting would not help reduce the generalization gap. The difference between the *training and testing* accuracies for the networks is due to the specific choice of the network (e.g. AlexNet, VGGNet etc.) and is not the focus of this study. Rather, our goal is to study the source of the testing performance disparity of the two regimes, SB and LB, on a given network model.



(a) Network F_2 (b) Network C_1

Figure 4.2. Training and testing accuracy for SB and LB methods as a function of epochs.

4.2.2.1. Parametric Plots. We first present parametric 1-D plots of the function as described in (Goodfellow et al., 2014b). Let $x_s^\star$ and $x_\ell^\star$ indicate the solutions obtained by running ADAM using small and large batch sizes respectively. We plot the loss function, on both training and testing data sets, along a line-segment containing the two points. Specifically, for $\alpha \in [-1, 2]$, we plot the function $f(\alpha x_\ell^\star + (1-\alpha)x_s^\star)$ and also superimpose the classification accuracy at the intermediate points; see Figure 4.3[1]. For this experiment, we randomly chose a pair of SB and LB minimizers from the 5 trials used to generate the data in Table 4.3. The plots show that the LB minima are strikingly sharper than the SB minima in this one-dimensional manifold. The plots in Figure 4.3 only explore a linear slice of the function, but in Figure 4.4, we plot $f(\sin(\frac{\alpha\pi}{2})x_\ell^\star + \cos(\frac{\alpha\pi}{2})x_s^\star)$ to monitor the function along a curved path between the two minimizers . There too, the relative sharpness of the minima is evident.

4.2.2.2. Sharpness of Minima. So far, we have used the term *sharp minimizer* loosely, but we noted that this concept has received attention in the literature (Hochreiter and Schmidhuber, 1997). Sharpness of a minimizer can be characterized by the magnitude of the eigenvalues of $\nabla^2 f(x)$, but given the prohibitive cost of this computation in deep learning applications, we employ a sensitivity measure that, although imperfect, is computationally feasible, even for large networks. It is based on exploring a small neighborhood of a solution and computing the largest value that the function f can attain in that neighborhood. We use that value to measure the sensitivity of the training function at the given local minimizer. Now, since the maximization process is not accurate, and to avoid being mislead by the case when a large value of f is attained only in a tiny subspace of $\mathbb{R}^n$, we

[1]The code to reproduce the parametric plot on exemplary networks can be found in our GitHub repository: `https://github.com/keskarnitish/large-batch-training`.

perform the maximization both in the entire space $\mathbb{R}^n$ as well as in random manifolds. For that purpose, we introduce an $n \times p$ matrix A, whose columns are randomly generated. Here p determines the dimension of the manifold, which in our experiments is chosen as $p = 100$.

Specifically, let $\mathcal{C}_\epsilon$ denote a box around the solution over which the maximization of f is performed, and let $A \in \mathbb{R}^{n\times p}$ be the matrix defined above. In order to ensure invariance of sharpness to problem dimension and sparsity, we define the constraint set $\mathcal{C}_\epsilon$ as:

$$\mathcal{C}_\epsilon = \{z \in \mathbb{R}^p : -\epsilon(|(A^+x)_i| + 1) \leq z_i \leq \epsilon(|(A^+x)_i| + 1) \quad \forall i \in \{1, 2, \cdots, p\}\}, \tag{4.3}$$

where A^+ denotes the pseudo-inverse of A. Thus ϵ controls the size of the box. We can now define our measure of sharpness (or sensitivity).

Metric 4.2.1. *Given $x \in \mathbb{R}^n$, $\epsilon > 0$ and $A \in \mathbb{R}^{n\times p}$, we define the $(\mathcal{C}_\epsilon, A)$-sharpness of f at x as:*

$$\phi_{x,f}(\epsilon, A) := \frac{(\max_{y\in\mathcal{C}_\epsilon} f(x + Ay)) - f(x)}{1 + f(x)} \times 100. \tag{4.4}$$

Unless specified otherwise, we use this metric for sharpness for the rest of the paper; if A is not specified, it is assumed to be the identity matrix, I_n. (We note in passing that, in the convex optimization literature, the term sharp minimum has a different definition (Ferris, 1988), but that concept is not useful for our purposes.)

In Tables 4.4 and 4.5, we present the values of the sharpness metric (4.4) for the minimizers of the various problems. Table 4.4 explores the full-space (i.e., $A = I_n$) whereas Table 4.5 uses a randomly sampled $n \times 100$ dimensional matrix A. We report results with

Table 4.4. Sharpness of Minima in Full Space; ϵ is defined in (4.3).

	$\epsilon = 10^{-3}$		$\epsilon = 5 \cdot 10^{-4}$	
	SB	LB	SB	LB
F_1	1.23 ± 0.83	205.14 ± 69.52	0.61 ± 0.27	42.90 ± 17.14
F_2	1.39 ± 0.02	310.64 ± 38.46	0.90 ± 0.05	93.15 ± 6.81
C_1	28.58 ± 3.13	707.23 ± 43.04	7.08 ± 0.88	227.31 ± 23.23
C_2	8.68 ± 1.32	925.32 ± 38.29	2.07 ± 0.86	175.31 ± 18.28
C_3	29.85 ± 5.98	258.75 ± 8.96	8.56 ± 0.99	105.11 ± 13.22
C_4	12.83 ± 3.84	421.84 ± 36.97	4.07 ± 0.87	109.35 ± 16.57

Table 4.5. Sharpness of Minima in Random Subspaces of Dimension 100

	$\epsilon = 10^{-3}$		$\epsilon = 5 \cdot 10^{-4}$	
	SB	LB	SB	LB
F_1	0.11 ± 0.00	9.22 ± 0.56	0.05 ± 0.00	9.17 ± 0.14
F_2	0.29 ± 0.02	23.63 ± 0.54	0.05 ± 0.00	6.28 ± 0.19
C_1	2.18 ± 0.23	137.25 ± 21.60	0.71 ± 0.15	29.50 ± 7.48
C_2	0.95 ± 0.34	25.09 ± 2.61	0.31 ± 0.08	5.82 ± 0.52
C_3	17.02 ± 2.20	236.03 ± 31.26	4.03 ± 1.45	86.96 ± 27.39
C_4	6.05 ± 1.13	72.99 ± 10.96	1.89 ± 0.33	19.85 ± 4.12

two values of ϵ, $(10^{-3}, 5 \cdot 10^{-4})$. In all experiments, we solve the maximization problem in Equation (4.4) inexactly by applying 10 iterations of L-BFGS-B (Byrd et al., 1995). This limit on the number of iterations was necessitated by the large cost of evaluating the true objective f. Both tables show a 1–2 order-of-magnitude difference between the values of our metric for the SB and LB regimes. These results reinforce the view that the solutions obtained by a large-batch method defines points of larger sensitivity of the training function. In Section 4.4, we describe approaches to attempt to remedy this generalization problem of LB methods. These approaches include data augmentation, conservative training and adversarial training. Our preliminary findings show that these approaches help reduce the generalization gap but still lead to relatively sharp minimizers and as such, do not completely remedy the problem.

Note that Metric 2.1 is closely related to the spectrum of $\nabla^2 f(x)$. Assuming ϵ to be small enough, when $A = I_n$, the value (4.4) relates to the largest eigenvalue of $\nabla^2 f(x)$ and when A is randomly sampled it approximates the Ritz value of $\nabla^2 f(x)$ projected onto the column-space of A.

We conclude this section by noting that the sharp minimizers identified in our experiments do not resemble a cone, i.e., the function does not increase rapidly along all (or even most) directions. By sampling the loss function in a neighborhood of LB solutions, we observe that it rises steeply only along a small dimensional subspace (e.g. 5% of the whole space); on most other directions, the function is relatively flat.

4.3. Success of Small-Batch Methods

It is often reported that when increasing the batch size for a problem, there exists a threshold after which there is a deterioration in the quality of the model. This behavior can be observed for the F_2 and C_1 networks in Figure 4.5. In both of these experiments, there is a batch size (≈ 15000 for F_2 and ≈ 500 for C_1) after which there is a large drop in testing accuracy. Notice also that the upward drift in value of the sharpness is considerably reduced around this threshold. Similar thresholds exist for the other networks in Table 4.1.

Let us now consider the behavior of SB methods, which use noisy gradients in the step computation. From the results reported in the previous section, it appears that noise in the gradient pushes the iterates out of the basin of attraction of sharp minimizers and encourages movement towards a flatter minimizer where noise will not cause exit from that basin. When the batch size is greater than the threshold mentioned above, the noise

in the stochastic gradient is not sufficient to cause ejection from the initial basin leading to convergence to sharper a minimizer.

To explore that in more detail, consider the following experiment. We train the network for 100 epochs using ADAM with a batch size of 256, and retain the iterate after each epoch in memory. Using these 100 iterates as starting points we train the network using a LB method for 100 epochs and receive a 100 *piggybacked* (or warm-started) large-batch solutions. We plot in Figure 4.6 the testing accuracy and sharpness of these large-batch solutions, along with the testing accuracy of the small-batch iterates. Note that when warm-started with only a few initial epochs, the LB method does not yield a generalization improvement. The concomitant sharpness of the iterates also stays high. On the other hand, after certain number of epochs of warm-starting, the accuracy improves and sharpness of the large-batch iterates drop. This happens, apparently, when the SB method has ended its exploration phase and discovered a flat minimizer; the LB method is then able to converge towards it, leading to good testing accuracy.

It has been speculated that LB methods tend to be attracted to minimizers close to the starting point x_0, whereas SB methods move away and locate minimizers that are farther away. Our numerical experiments support this view: we observed that the ratio of $\|x_s^\star - x_0\|_2$ and $\|x_\ell^\star - x_0\|_2$ was in the range of 3–10.

In order to further illustrate the qualitative difference between the solutions obtained by SB and LB methods, we plot in Figure 4.7 our sharpness measure (4.4) against the loss function (cross entropy) for one random trial of the F_2 and C_1 networks. For larger values of the loss function, i.e., near the initial point, SB and LB method yield similar values of sharpness. As the loss function reduces, the sharpness of the iterates corresponding

to the LB method rapidly increases, whereas for the SB method the sharpness stays relatively constant initially and then reduces, suggesting an exploration phase followed by convergence to a flat minimizer.

4.4. Attempts to Improve LB Methods

In this section, we discuss a few strategies that aim to remedy the problem of poor generalization for large-batch methods. As in Section 4.2, we use 10% as the percentage batch-size for large-batch experiments and 256 for small-batch methods. For all experiments, we use ADAM as the optimizer irrespective of batch-size.

4.4.1. Data Augmentation

Given that large-batch methods appear to be attracted to sharp minimizers, one can ask whether it is possible to modify the geometry of the loss function so that it is more benign to large-batch methods. The loss function depends both on the geometry of the objective function and to the size and properties of the training set. One approach we consider is data augmentation; see e.g. (Krizhevsky et al., 2012; Simonyan and Zisserman, 2014). The application of this technique is domain specific but generally involves augmenting the data set through controlled modifications on the training data. For instance, in the case of image recognition, the training set can be augmented through translations, rotations, shearing and flipping of the training data. This technique leads to regularization of the network and has been employed for improving testing accuracy on several data sets.

In our experiments, we train the 4 image-based (convolutional) networks using aggressive data augmentation and present the results in Table 4.6. For the augmentation, we use

Table 4.6. Effect of Data Augmentation

	Testing Accuracy		Sharpness (LB method)	
	Baseline (SB)	Augmented LB	$\epsilon = 10^{-3}$	$\epsilon = 5 \cdot 10^{-4}$
C_1	$83.63\% \pm 0.14\%$	$82.50\% \pm 0.67\%$	231.77 ± 30.50	45.89 ± 3.83
C_2	$89.82\% \pm 0.12\%$	$90.26\% \pm 1.15\%$	468.65 ± 47.86	105.22 ± 19.57
C_3	$54.55\% \pm 0.44\%$	$53.03\% \pm 0.33\%$	103.68 ± 11.93	37.67 ± 3.46
C_4	$63.05\% \pm 0.5\%$	$65.88 \pm 0.13\%$	271.06 ± 29.69	45.31 ± 5.93

horizontal reflections, random rotations up to 10° and random translation of up to 0.2 times the size of the image. It is evident from the table that, while the LB method achieves accuracy comparable to the SB method (also with training data augmented), the sharpness of the minima still exists, suggesting sensitivity to images contained in neither training or testing set. In this section, we exclude parametric plots and sharpness values for the SB method owing to space constraints and the similarity to those presented in Section 4.2.2.

4.4.2. Conservative Training

In (Li et al., 2014), the authors argue that the convergence rate of SGD for the large-batch setting can be improved by obtaining iterates through the following proximal sub-problem.

$$x_{k+1} = \arg\min_x \frac{1}{|B_k|} \sum_{i \in B_k} f_i(x) + \frac{\lambda}{2} \|x - x_k\|_2^2 \tag{4.5}$$

The motivation for this strategy is, in the context of large-batch methods, to better utilize a batch before moving onto the next one. The minimization problem is solved inexactly using 3–5 iterations of gradient descent, co-ordinate descent or L-BFGS. (Li et al., 2014) report that this not only improves the convergence rate of SGD but also leads to improved empirical performance on convex machine learning problems. The underlying idea of utilizing a batch is not specific to convex problems and we can apply the same framework

Table 4.7. Effect of Conservative Training

	Testing Accuracy		Sharpness (LB method)	
	Baseline (SB)	Conservative LB	$\epsilon = 10^{-3}$	$\epsilon = 5 \cdot 10^{-4}$
F_1	$98.03\% \pm 0.07\%$	$98.12\% \pm 0.01\%$	232.25 ± 63.81	46.02 ± 12.58
F_2	$64.02\% \pm 0.2\%$	$61.94\% \pm 1.10\%$	928.40 ± 51.63	190.77 ± 25.33
C_1	$80.04\% \pm 0.12\%$	$78.41\% \pm 0.22\%$	520.34 ± 34.91	171.19 ± 15.13
C_2	$89.24\% \pm 0.05\%$	$88.495\% \pm 0.63\%$	632.01 ± 208.01	108.88 ± 47.36
C_3	$49.58\% \pm 0.39\%$	$45.98\% \pm 0.54\%$	337.92 ± 33.09	110.69 ± 3.88
C_4	$63.08\% \pm 0.10\%$	62.51 ± 0.67	354.94 ± 20.23	68.76 ± 16.29

for deep learning, however, without theoretical guarantees. Indeed, similar algorithms were proposed in (Zhang et al., 2015) and (Mobahi, 2016) for Deep Learning. The former placed emphasis on parallelization of small-batch SGD and asynchrony while the latter on a diffusion-continuation mechanism for training. The results using the conservative training approach are presented in Figure 4.7. In all experiments, we solve the problem (4.5) using 3 iterations of ADAM and set the regularization parameter λ to be 10^{-3}. Again, there is a statistically significant improvement in the testing accuracy of the large-batch method but it does not solve the problem of sensitivity.

4.4.3. Robust Training

A natural way of avoiding sharp minima is through *robust optimization* techniques. These methods attempt to optimize a worst-case cost as opposed to the nominal (or true) cost. Mathematically, given an $\epsilon > 0$, these techniques solve the problem

$$\min_x \quad \phi(x) := \max_{\|\Delta x\| \leq \epsilon} f(x + \Delta x) \tag{4.6}$$

Geometrically, classical (nominal) optimization attempts to locate the lowest point of a valley, while robust optimization attempts to lower an ϵ–disc down the loss surface.

We refer an interested reader to (Bertsimas et al., 2010), and the references therein, for a review of non-convex robust optimization. A direct application of this technique is, however, not feasible in our context since each iteration is prohibitively expensive because it involves solving a large-scale second-order conic program (SOCP).

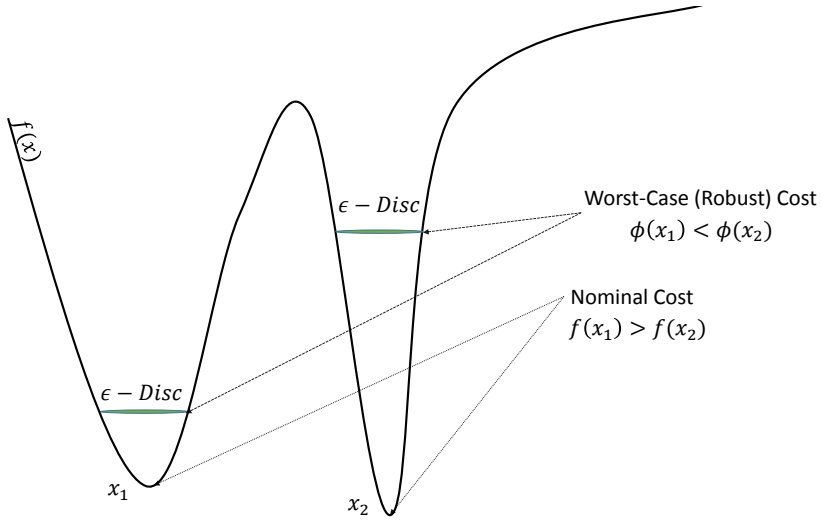


Figure 4.8. Illustration of Robust Optimization

In the context of Deep Learning, there are two inter-dependent forms of robustness: robustness to the data and robustness to the solution. The former exploits the fact that the function f is inherently a statistical model, while the latter treats f as a black-box function. In (Shaham et al., 2015), the authors prove the equivalence between robustness of the solution (with respect to the data) and adversarial training (Goodfellow et al., 2014a).

Given the partial success of the data augmentation strategy, it is natural to question the efficacy of adversarial training. As described in (Goodfellow et al., 2014a), adversarial training also aims to artificially increase the training set but, unlike randomized data augmentation, uses the model's sensitivity to construct new examples. Despite its intuitive

appeal, in our experiments, we found that this strategy did not improve generalization. Similarly, we observed no generalization benefit from the stability training proposed by (Zheng et al., 2016). In both cases, the testing accuracy, sharpness values and the parametric plots were similar to the unmodified (baseline) case discussed in Section 4.2. It remains to be seen whether adversarial training (or any other form of robust training) can increase the viability of large-batch training.

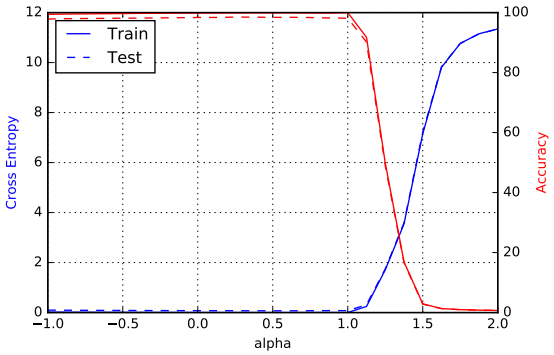
4.5. Discussion and Conclusion

In this paper, we present numerical experiments that support the view that convergence to sharp minimizers gives rise to the poor generalization of large-batch methods for deep learning. To this end, we provide one-dimensional parametric plots and perturbation (sharpness) measures for a variety of deep learning architectures. In Section 4.4, we describe our attempts to remedy the problem, including data augmentation, conservative training and robust optimization. Our preliminary investigation suggests that these strategies do not correct the problem; they improve the generalization of large-batch methods but still lead to relatively sharp minima. Another prospective remedy includes the use of *dynamic sampling* where the batch size is increased gradually as the iteration progresses (Byrd et al., 2012b; Friedlander and Schmidt, 2012). The potential viability of this approach is suggested by our warm-starting experiments (see Figure 4.6) wherein high testing accuracy is achieved using a large-batch method that is warm-start with a small-batch method.

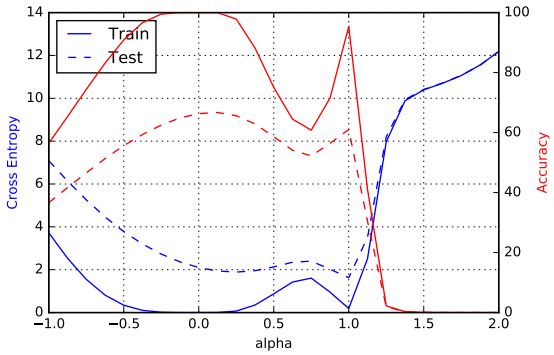
Recently, a number of researchers have described interesting theoretical properties of the loss surface of deep neural networks; see e.g. (Choromanska et al., 2015; Soudry and Carmon, 2016; Lee et al., 2016). Their work shows that, under certain regularity

assumptions, the loss function of deep learning models is fraught with many local minimizers and that many of these minimizers correspond to a similar loss function value. Our results are in alignment these observations since, in our experiments, both sharp and flat minimizers have very similar loss function values. We do not know, however, if the theoretical models mentioned above provide information about the existence and density of sharp minimizers of the loss surface.

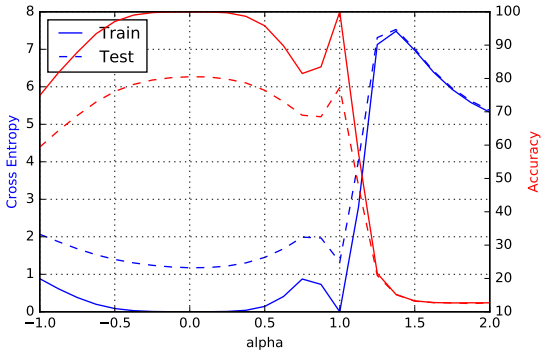
Our results suggest some questions: (a) can one *prove* that large-batch (LB) methods typically converge to sharp minimizers of deep learning training functions? (In this paper, we only provided some numerical evidence.); (b) what is the relative density of the two kinds of minima?; (c) can one design neural network architectures for various tasks that are suitable to the properties of LB methods?; (d) can the networks be initialized in a way that enables LB methods to succeed?; (e) is it possible, through algorithmic or regulatory means to steer LB methods away from sharp minimizers?



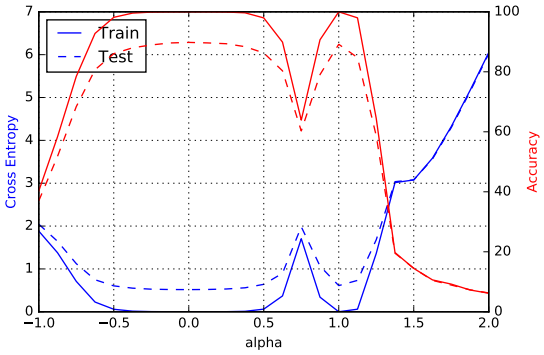
(a) F_1



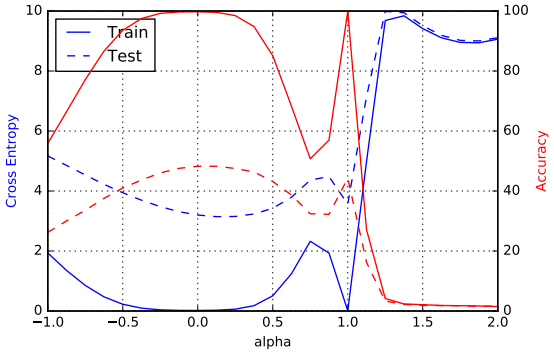
(b) F_2



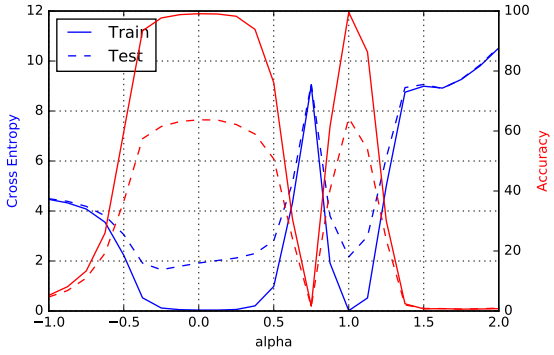
(c) C_1



(d) C_2

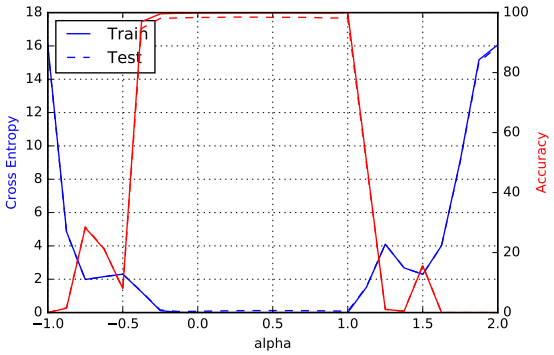


(e) C_3

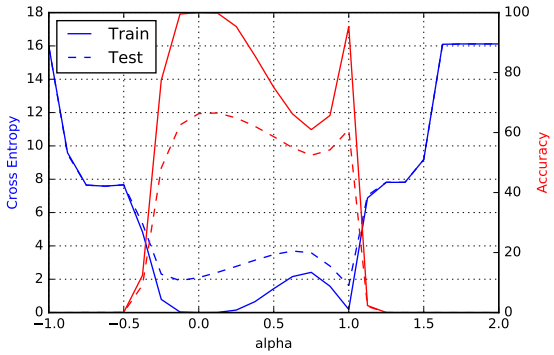


(f) C_4

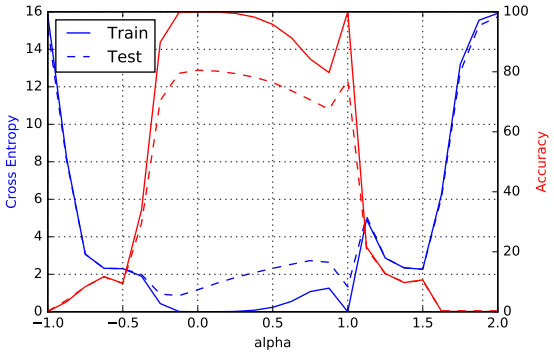
Figure 4.3. Parametric Plots – Linear (Left vertical axis corresponds to cross-entropy loss, f, and right vertical axis corresponds to classification accuracy; solid line indicates training data set and dashed line indicated testing data set); $\alpha = 0$ corresponds to the SB minimizer and $\alpha = 1$ to the LB minimizer.



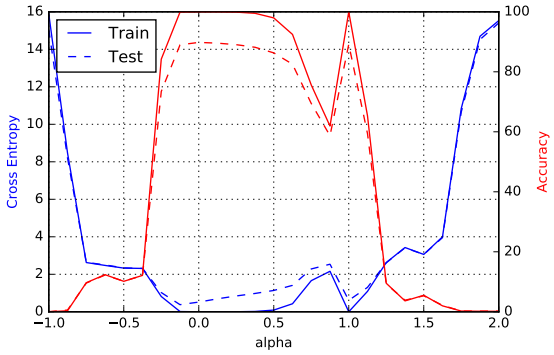
(a) F_1



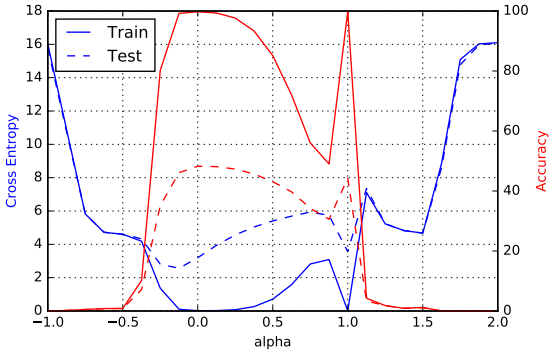
(b) F_2



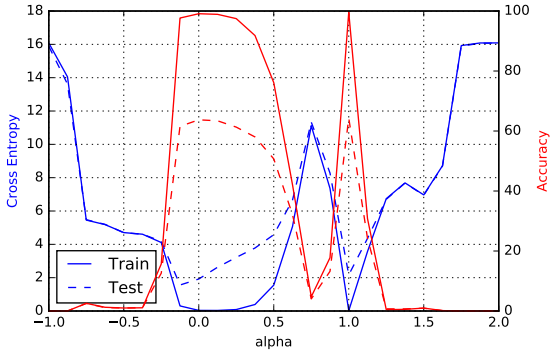
(c) C_1



(d) C_2

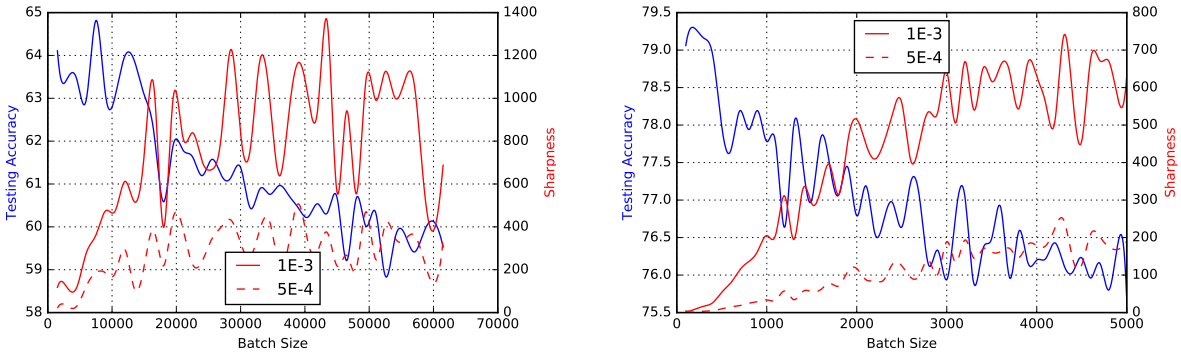


(e) C_3



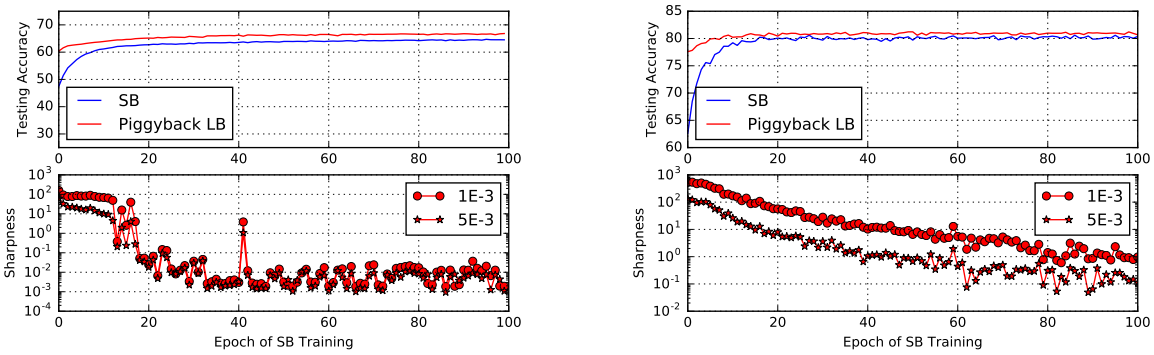
(f) C_4

Figure 4.4. Parametric Plots – Curvilinear (Left vertical axis corresponds to cross-entropy loss, f, and right vertical axis corresponds to classification accuracy; solid line indicates training data set and dashed line indicated testing data set); $\alpha = 0$ corresponds to the SB minimizer while $\alpha = 1$ corresponds to the LB minimizer



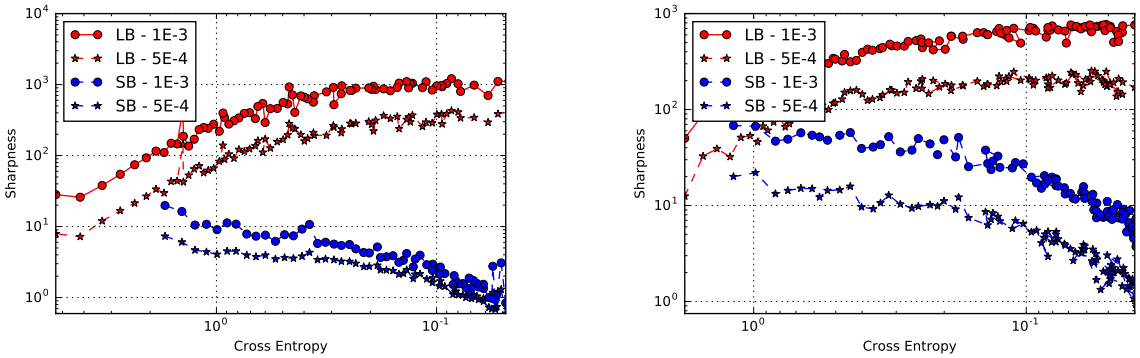
(a) F_2 (b) C_1

Figure 4.5. Testing Accuracy and Sharpness v/s Batch Size. The X-axis corresponds to the batch size used for training the network for 100 epochs, left Y-axis corresponds to the testing accuracy at the final iterate and right Y-axis corresponds to the sharpness of that iterate. We report sharpness for two values of ϵ: 10^{-3} and $5 \cdot 10^{-4}$.



(a) F_2 (b) C_1

Figure 4.6. Warm-starting experiments. The upper figures report the testing accuracy of the SB method (blue line) and the testing accuracy of the warm started (piggybacked) LB method (red line), as a function of the number of epochs of the SB method. The lower figures plot the sharpness measure (4.4) for the solutions obtained by the piggybacked LB method v/s the number of warm-starting epochs of the SB method.



(a) F_2 (b) C_1

Figure 4.7. Sharpness v/s Cross Entropy Loss for SB and LB methods.

References

Andersen, M. S., J. Dahl, L. Vandenberghe. 2013. CVXOPT: A Python package for convex optimization, version 1.1.8. *Available at cvxopt.org* .

Andrew, G., J. Gao. 2007. Scalable training of L_1-regularized log-linear models. *Proceedings of the 24th international conference on Machine Learning*. ACM, 33–40.

Bach, F., R. Jenatton, J. Mairal, G. Obozinski. 2012. Optimization with sparsity-inducing penalties. *Foundations and Trends in Machine Learning* **4**(1) 1–106.

Bagirov, A., N. Karmitsa, M. M. Mäkelä. 2014. *Introduction to nonsmooth optimization: Theory, practice and software*. Springer.

Beck, A., M. Teboulle. 2009. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM Journal on Imaging Sciences* **2**(1) 183–202.

Bengio, Y., I. Goodfellow, A. Courville. 2016. Deep learning. URL `http://www.deeplearningbook.org`. Book in preparation for MIT Press.

Bertsekas, D. P. 1982. Projected Newton methods for optimization problems with simple constraints. *SIAM Journal on control and Optimization* **20**(2) 221–246.

Bertsimas, D., O. Nohadani, Kwong M. Teo. 2010. Robust optimization for unconstrained simulation-based problems. *Operations Research* **58**(1) 161–178.

Bottou, L. 1998. Online learning and stochastic approximations. *On-line learning in neural networks* **17**(9) 142.

Bottou, L., F. E. Curtis, J. Nocedal. 2016. Optimization methods for large-scale machine learning. *arXiv preprint arXiv:1606.04838* .

Boyd, S. P., L. Vandenberghe. 2004. *Convex optimization*. Cambridge Univ Pr.

Burke, J. V., A. S. Lewis, M. L. Overton. 2005. A robust gradient sampling algorithm for nonsmooth, nonconvex optimization. *SIAM Journal on Optimization* **15**(3) 751–779.

Byrd, R. H., G. M. Chin, J. Nocedal, F. Oztoprak. 2012a. A family of second-order methods for convex L1 regularized optimization. Tech. rep., Optimization Center Report 2012/2, Northwestern University.

Byrd, R. H., G. M. Chin, J. Nocedal, F. Oztoprak. 2016. A family of second-order methods for convex ℓ_1-regularized optimization. *Mathematical Programming* **159**(1) 435–467. doi: 10.1007/s10107-015-0965-3. URL `http://dx.doi.org/10.1007/s10107-015-0965-3`.

Byrd, R. H., G. M. Chin, J. Nocedal, Y. Wu. 2012b. Sample size selection in optimization methods for machine learning. *Mathematical Programming* **134**(1) 127–155.

Byrd, R. H., P. Lu, J. Nocedal, C. Zhu. 1995. A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing* **16**(5) 1190–1208.

Byrd, R. H., J. Nocedal, F. Oztoprak. 2015. An inexact successive quadratic approximation method for l-1 regularized optimization. *Mathematical Programming* 1–22doi:10.1007/s10107-015-0941-y. URL `http://dx.doi.org/10.1007/s10107-015-0941-y`.

Byrd, R. H., J. Nocedal, R. Schnabel. 1994a. Representations of quasi-Newton matrices and their use in limited memory methods. *Mathematical Programming* **63**(4) 129–156.

Byrd, R. H., J. Nocedal, R. B. Schnabel. 1994b. Representations of quasi-Newton matrices and their use in limited memory methods. *Mathematical Programming* **63**(1) 129–156.

Chaudhari, P., A. Choromanska, S. Soatto, Y. LeCun. 2016. Entropy-SGD: Biasing gradient descent into wide valleys. *arXiv preprint arXiv:1611.01838* .

Choromanska, A., M. Henaff, M. Mathieu, G. Arous, Y. LeCun. 2015. The loss surfaces of multilayer networks. *AISTATS*.

Clarke, F. H. 1990. *Optimization and nonsmooth analysis*, vol. 5. SIAM.

Curtis, F. E., Z. Han, D. P. Robinson. 2015. A globally convergent primal-dual active-set framework for large-scale convex quadratic optimization. *Computational Optimization and Applications* **60**(2) 311–341.

Curtis, F. E., M. L. Overton. 2012. A sequential quadratic programming algorithm for nonconvex, nonsmooth constrained optimization. *SIAM Journal on Optimization* **22**(2) 474–500.

Curtis, F. E., X. Que. 2013. An adaptive gradient sampling algorithm for non-smooth optimization. *Optimization Methods and Software* **28**(6) 1302–1324.

Curtis, F. E., X. Que. 2015. A quasi-Newton algorithm for nonconvex, nonsmooth optimization with global convergence guarantees. *Mathematical Programming Computation* **7**(4) 399–428.

Das, D., S. Avancha, D. Mudigere, K. Vaidynathan, S. Sridharan, D. Kalamkar, B. Kaul, P. Dubey. 2016. Distributed deep learning using synchronous stochastic gradient descent. *arXiv preprint arXiv:1602.06709* .

Daubechies, I., M. Defrise, C. De Mol. 2004. An iterative thresholding algorithm for linear inverse problems with a sparsity constraint. *Communications on Pure and Applied Mathematics* **57**(11) 1413–1457.

De Santis, M., S. Lucidi, F. Rinaldi. 2014. A fast active set block coordinate descent algorithm for ℓ_1-regularized least squares. *arXiv preprint arXiv:1403.1738* .

Dean, J., G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, A. Senior, P. Tucker, K. Yang, Q. V. Le, et al. 2012. Large scale distributed deep networks. *Advances in neural information processing systems*. 1223–1231.

Dolan, E. D., J. J. Moré. 2002. Benchmarking optimization software with performance profiles. *Mathematical programming* **91**(2) 201–213.

Donoho, D.L. 1995. De-noising by soft-thresholding. *Information Theory, IEEE Transactions on* **41**(3) 613–627.

Duchi, J., E. Hazan, Y. Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research* **12** 2121–2159.

Ferris, M. 1988. Weak sharp minima and penalty functions in mathematical programming. Ph.D. thesis, University of Cambridge.

Ferry, M. W. 2011. Projected-search methods for box-constrained optimization. Ph.D. thesis, Department of Mathematics, University of California at San Diego.

Fountoulakis, K., J. Gondzio, P. Zhlobich. 2014. Matrix-free interior point method for compressed sensing problems. *Mathematical Programming Computation* **6**(1) 1–31.

Fountoulakis, Kimon, Jacek Gondzio. 2015. Performance of first-and second-order methods for big data optimization. *arXiv preprint arXiv:1503.03520* .

Friedlander, M. P., M. Schmidt. 2012. Hybrid deterministic-stochastic methods for data fitting. *SIAM Journal on Scientific Computing* **34**(3) A1380–A1405.

Friedman, J., T. Hastie, R. Tibshirani. 2001. *The elements of statistical learning*, vol. 1. Springer series in statistics Springer, Berlin.

Friedman, J., T. Hastie, R. Tibshirani. 2010. Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software* **33**(1) 1. URL `http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2929880/`.

Garofolo, J. S., L. F. Lamel, W. M. Fisher, J. G. Fiscus, D. S. Pallett, N. L. Dahlgren, V. Zue. 1993. Timit acoustic-phonetic continuous speech corpus. *Linguistic data consortium, Philadelphia* **33**.

Ge, R., F. Huang, C. Jin, Y. Yuan. 2015. Escaping from saddle points online stochastic gradient for tensor decomposition. *Proceedings of The 28th Conference on Learning Theory*. 797–842.

Goodfellow, I., J. Shlens, C. Szegedy. 2014a. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572* .

Goodfellow, Ian J, Oriol Vinyals, Andrew M Saxe. 2014b. Qualitatively characterizing neural network optimization problems. *arXiv preprint arXiv:1412.6544* .

Graves, A., A. Mohamed, G. Hinton. 2013. Speech recognition with deep recurrent neural networks. *2013 IEEE international conference on acoustics, speech and signal processing*. IEEE, 6645–6649.

Greene, D., P. Cunningham. 2006. Practical solutions to the problem of diagonal dominance in kernel document clustering. *Proceedings of the 23rd International Conference on Machine Learning*. ICML '06, ACM, New York, NY, USA, 377–384. doi:10.1145/1143844.1143892. URL `http://doi.acm.org/10.1145/1143844.1143892`.

Gürbüzbalaban, M., M. L. Overton. 2012. On Nesterovs nonsmooth Chebyshev–Rosenbrock functions. *Nonlinear Analysis: Theory, Methods and Applications* **75**(3) 1282–1289.

Haarala, M., K. Miettinen, M. M. Mäkelä. 2004. New limited memory bundle method for large-scale nonsmooth optimization. *Optimization Methods and Software* **19**(6) 673–692. doi:10.1080/10556780410001689225. URL `http://dx.doi.org/10.1080/10556780410001689225`.

Haarala, N., K. Miettinen, M. M. Mäkelä. 2007. Globally convergent limited memory bundle method for large-scale nonsmooth optimization. *Mathematical Programming* **109**(1) 181–205.

Han, Z., F. E. Curtis. 2015. Primal-dual active-set methods for isotonic regression and trend filtering. *arXiv preprint arXiv:1508.02452* .

Hardt, M., B. Recht, Y. Singer. 2015. Train faster, generalize better: Stability of stochastic gradient descent. *arXiv preprint arXiv:1509.01240* .

Henao, W. 2014. An L-BFGS-B-NS optimizer for non-smooth functions. Master's thesis, Courant Institute of Mathematical Science, New York University.

Hochreiter, Sepp, Jürgen Schmidhuber. 1997. Flat minima. *Neural Computation* **9**(1) 1–42.

Homem-de Mello, T., G. Bayraksan. 2014. Monte carlo sampling-based methods for stochastic optimization. *Surveys in Operations Research and Management Science* **19**(1) 56–85.

Hsieh, C. J., M. A. Sustik, P. Ravikumar, I. S. Dhillon. 2011. Sparse inverse covariance matrix estimation using quadratic approximation. *Advances in Neural Information Processing Systems (NIPS)* **24** 2330–2338.

Hungerländer, P., F. Rendl. 2015. A feasible active set method for strictly convex quadratic problems with simple bounds. *SIAM Journal on Optimization* **25**(3) 1633–1659.

Ioffe, S., C. Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167* .

J., Lee, Yuekai S., Saunders M. 2014. Proximal newton-type methods for minimizing composite functions. *SIAM Journal on Optimization* **24**(3) 1420–1443. doi:10.1137/130921428. URL `http://dx.doi.org/10.1137/130921428`.

Kaku, A. 2011. Implementation of high precision arithmetic in the BFGS method for nonsmooth optimization. Master's thesis, Courant Institute of Mathematical Science, New York University.

Karmitsa, N., A. Bagirov, M. M. Mäkelä. 2012. Comparing different nonsmooth minimization methods and software. *Optimization Methods and Software* **27**(1) 131–153. doi:10.1080/10556788.2010.526116. URL `http://dx.doi.org/10.1080/10556788.2010.526116`.

Karmitsa, N., M. M. Mäkelä. 2010a. Adaptive limited memory bundle method for bound constrained large-scale nonsmooth optimization. *Optimization* **59**(6) 945–962.

Karmitsa, N., M. M. Mäkelä. 2010b. Limited memory bundle method for large bound constrained nonsmooth optimization: Convergence analysis. *Optimization Methods and Software* **25**(6) 895–916.

Keskar, N., A. S. Berahas. 2016. *adaQN: An Adaptive Quasi-Newton Algorithm for Training RNNs*. Springer International Publishing, Cham, 1–16.

Keskar, N., J. Nocedal, F. Öztoprak, A. Wächter. 2016. A second-order method for convex ℓ_1-regularized optimization with active-set prediction. *Optimization Methods and Software* **31**(3) 605–621.

Kingma, D., J. Ba. 2015. Adam: A method for stochastic optimization. *International Conference on Learning Representations (ICLR 2015)*.

Kiwiel, K. C. 2007. Convergence of the gradient sampling algorithm for nonsmooth nonconvex optimization. *SIAM Journal on Optimization* **18**(2) 379–388.

Koh, K., S. Kim, S. P. Boyd. 2007. An interior-point method for large-scale l1-regularized logistic regression. *Journal of Machine learning research* **8**(8) 1519–1555.

Krizhevsky, A., G. E. Hinton. 2009. Learning multiple layers of features from tiny images .

Krizhevsky, A., I. Sutskever, G. E. Hinton. 2012. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*. 1097–1105.

LeCun, Y., L. Bottou, Y. Bengio, P. Haffner. 1998a. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* **86**(11) 2278–2324.

LeCun, Y., L. Bottou, G. B. Orr, K. Müller. 2012. Efficient backprop. *Neural networks: Tricks of the trade*. Springer, 9–48.

LeCun, Y., C. Cortes, C. Burges. 1998b. The mnist database of handwritten digits.

Lee, J., M. Simchowitz, M. Jordan, B. Recht. 2016. Gradient descent converges to minimizers. *University of California, Berkeley* **1050** 16.

Lewis, A. S., M. L. Overton. 2013. Nonsmooth optimization via quasi-Newton methods. *Mathematical Programming* **141**(1) 135–163. doi:10.1007/s10107-012-0514-2. URL `http://dx.doi.org/10.1007/s10107-012-0514-2`.

Lewis, A. S., S. Zhang. 2015. Nonsmoothness and a variable metric method. *Journal of Optimization Theory and Applications* **165**(1) 151–171. doi:10.1007/s10957-014-0622-7. URL `http://dx.doi.org/10.1007/s10957-014-0622-7`.

Li, M., T. Zhang, Y. Chen, A. J. Smola. 2014. Efficient mini-batch training for stochastic optimization. *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 661–670.

Liu, D. C., J. Nocedal. 1989. On the limited memory BFGS method for large scale optimization. *Mathematical programming* **45**(1-3) 503–528.

Lukšan, L., M. Tuma, J. Vlcek, N. Ramešová, M. Šiška, J. Hartman, C. Matonoha. 2014. UFO 2004 - interactive system for universal functional optimization. Tech. Rep. 1218, Institute of Computer Science, Academy of Science of the Czech Republic.

MacKay, D. 1992. A practical bayesian framework for backpropagation networks. *Neural computation* **4**(3) 448–472.

Mäkelä, M. M. 2002. Survey of bundle methods for nonsmooth optimization. *Optimization Methods and Software* **17** 1.

Mnih, V., K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M. Riedmiller. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* .

Mobahi, H. 2016. Training recurrent neural networks by diffusion. *arXiv preprint arXiv:1601.04114* .

Nesterov, Y. 2012. Efficiency of coordinate descent methods on huge-scale optimization problems. *SIAM Journal on Optimization* **22**(2) 341–362.

Nocedal, J., S. J. Wright. 2006. *Numerical optimization*. 2nd ed. Springer, New York.

Olsen, P., F. Oztoprak, J. Nocedal, S. Rennie. 2012. Newton-like methods for sparse inverse covariance estimation. P. Bartlett, F.c.n. Pereira, C.j.c. Burges, L. Bottou, K.q. Weinberger, eds., *Advances in Neural Information Processing Systems 25*. 764–772. URL `http://books.nips.cc/papers/files/nips25/NIPS2012_0344.pdf`.

Overton, M. 02/23/2016. private communication.

Pascal Large Scale Learning Challenge. 2008. Pascal large scale learning challenge. `http://largescale.ml.tu-berlin.de/`. Accessed: 2015-01-01.

Povey, D., A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlicek, Y. Qian, P. Schwarz, et al. 2011. The kaldi speech recognition toolkit. *IEEE 2011 workshop on automatic speech recognition and understanding*. EPFL-CONF-192584, IEEE Signal Processing Society.

Richtárik, P., M. Takáč. 2014. Iteration complexity of randomized block-coordinate descent methods for minimizing a composite function. *Mathematical Programming* **144**(1-2) 1–38.

Rissanen, J. 1983. A universal prior for integers and estimation by minimum description length. *The Annals of statistics* 416–431.

Scheinberg, K., X. Tang. 2014. Practical inexact proximal quasi-Newton method with global complexity analysis. *arXiv preprint arXiv:1311.6547* .

Schmidt, M. 2010. Graphical model structure learning with l1-regularization. Ph.D. thesis, University of British Columbia.

Schmidt, M., G. Fung, R. Rosales. 2007. Fast optimization methods for l1 regularization: A comparative study and two new approaches. *Machine Learning: ECML 2007*. Springer, 286–297.

Schmidt, M., D. Kim, S. Suvrit. 2011. Projected Newton-type methods in machine learning. *Optimization for Machine Learning* .

Shaham, U., Y. Yamada, S. Negahban. 2015. Understanding adversarial training: Increasing local stability of neural nets through robust optimization. *arXiv preprint arXiv:1511.05432* .

Simonyan, K., A. Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* .

Skajaa, A. 2010. Limited memory BFGS for nonsmooth optimization. Master's thesis, Courant Institute of Mathematical Science, New York University.

Solntsev, S., J. Nocedal, R. H. Byrd. 2014. An algorithm for quadratic 1-regularized optimization with a flexible active-set strategy. *Optimization Methods and Software* (ahead-of-print) 1–25.

Soudry, D., Y. Carmon. 2016. No bad local minima: Data independent training error guarantees for multilayer neural networks. *arXiv preprint arXiv:1605.08361* .

Sra, S., S. Nowozin, S.J. Wright. 2011. *Optimization for Machine Learning*. Mit Press.

Srivastava, N., G. E. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research* **15**(1) 1929–1958.

Sutskever, I., J. Martens, G. Dahl, G. Hinton. 2013. On the importance of initialization and momentum in deep learning. *Proceedings of the 30th International Conference on Machine Learning (ICML 2013)*. 1139–1147.

Team, The Theano Development, R. Al-Rfou, G. Alain, A. Almahairi, C. Angermueller, D. Bahdanau, N. Ballas, F. Bastien, J. Bayer, A. Belikov, et al. 2016. Theano: A Python framework for fast computation of mathematical expressions. *arXiv preprint arXiv:1605.02688* .

Tseng, P., S. Yun. 2009. A coordinate gradient descent method for nonsmooth separable minimization. *Mathematical Programming* **117**(1-2) 387–423.

Walt, S., S. C. Colbert, G. Varoquaux. 2011. The NumPy array: A structure for efficient numerical computation. *Computing in Science and Engineering* **13**(2) 22–30. doi:http://dx.doi.org/10.1109/MCSE.2011.37. URL `http://scitation.aip.org/content/aip/journal/cise/13/2/10.1109/MCSE.2011.37`.

Wen, Z., W. Yin, D. Goldfarb, Y. Zhang. 2010. A fast algorithm for sparse reconstruction based on shrinkage, subspace optimization and continuation. *SIAM Journal on Scientific Computing* **32**(4) 1832–1857.

Wright, S. 2014. Coordinate descent algorithms. Technical report, University of Wisconsin, Madison, Wisconsin, U.S.A.

Wright, S.J., R.D. Nowak, M.A.T. Figueiredo. 2009. Sparse reconstruction by separable approximation. *IEEE Transactions on Signal Processing* **57**(7) 2479–2493.

Yuan, G-X., C-H. Ho, C-J. Lin. 2012. An improved glmnet for l1-regularized logistic regression. *The Journal of Machine Learning Research* **13**(1) 1999–2030.

Zhang, S., A. Choromanska, Y. LeCun. 2015. Deep learning with elastic averaging sgd. *Advances in Neural Information Processing Systems*. 685–693.

Zheng, S., Y. Song, T. Leung, I. Goodfellow. 2016. Improving the robustness of deep neural networks via stability training. *arXiv preprint arXiv:1604.04326* .

APPENDIX A

Network Architecture and Performance Model Details

A.1. Architecture of Networks

A.1.1. Network F_1

For this network, we use a 784-dimensional input layer followed by 5 batch-normalized (Ioffe and Szegedy, 2015) layers of 512 neurons each with `ReLU` activations. The output layer consists of 10 neurons with the softmax activation.

A.1.2. Network F_2

The network architecture for F_2 is similar to F_1. We use a 360-dimensional input layer followed by 7 batch-normalized layers of 512 neurons with `ReLU` activation. The output layer consists of 1973 neurons with the softmax activation.

A.1.3. Networks C_1 and C_3

The C_1 network is a modified version of the popular AlexNet configuration (Krizhevsky et al., 2012). For simplicity, denote a stack of n convolution layers of a filters and a Kernel size of $b \times c$ with stride length of d as $n \times [a, b, c, d]$. The C_1 configuration uses 2 sets of $[64, 5, 5, 2]$–MaxPool(3) followed by 2 dense layers of sizes $(384, 192)$ and finally, an output layer of size 10. We use batch-normalization for all layers and `ReLU` activations. We also

use Dropout (Srivastava et al., 2014) of 0.5 retention probability for the two dense layers. The configuration C_3 is identical to C_1 except it uses 100 softmax outputs instead of 10.

A.1.4. Networks C_2 and C_4

The C_2 network is a modified version of the popular VGG configuration (Simonyan and Zisserman, 2014). The C_3 network uses the configuration: $2\times[64, 3, 3, 1], 2\times[128, 3, 3, 1], 3\times[256, 3, 3, 1], 3 \times [512, 3, 3, 1], 3 \times [512, 3, 3, 1]$ which a MaxPool(2) after each stack. This stack is followed by a 512-dimensional dense layer and finally, a 10-dimensional output layer. The activation and properties of each layer is as in A.1.3. As is the case with C_3 and C_1, the configuration C_4 is identical to C_2 except that it uses 100 softmax outputs instead of 10.

A.2. Performance Model

As mentioned in Section 4.1, a training algorithm that operates in the large-batch regime without suffering from a generalization gap would have the ability to scale to much larger number of nodes than is currently possible. Such and algorithm might also improve training time through faster convergence. We present an idealized performance model that demonstrates our goal.

For LB method to be competitive with SB method, the LB method must (i) converge to minimizers that generalize well, and (ii) do it in a reasonably number of iterations, which we analyze here. Let I_s and I_ℓ be number of iterations required by SB and LB methods to reach the point of comparable test accuracy, respectively. Let B_s and B_ℓ be corresponding batch sizes and P be number of processors being used for training. Assume

that $P < B_\ell$, and let $f_s(P)$ be the parallel efficiency of the SB method. For simplicity, we assume that $f_\ell(P)$, the parallel efficiency of the LB method, is 1.0. In other words, we assume that the LB method is perfectly scalable due to use of a large batch size.

For LB to be faster than SB, we must have

$$I_\ell \frac{B_\ell}{P} < I_s \frac{B_s}{P f_s(P)}.$$

In other words, the ratio of iterations of LB to the iterations of SB should be

$$\frac{I_\ell}{I_s} < \frac{B_s}{f_s(P) B_\ell}.$$

For example, if $f_s(P) = 0.2$ and $B_s/B_\ell = 0.1$, the LB method must converge in at most half as many iterations as the SB method to see performance benefits. We refer the reader to (Das et al., 2016) for a more detailed model and a commentary on the effect of batch-size on the performance.