# New Quasi-Newton Optimization Methods for Machine Learning

**Jin Yu**

A thesis submitted for the degree of
Doctor of Philosophy of
The Australian National University

September 2009

*To my parents, Zuohuang Yu and Zenyu Chen,*
*and my husband, Dan Liu,*
*for their unconditional love and support.*

# Declaration

Except where otherwise indicated, this thesis is my own original work.

This thesis is in part based on the following publications completed during my candidature: Chapters 3 and 4 are based on (Yu et al., 2008a,b); Chapters 5 and 6 are based on (Schraudolph et al., 2007). I was the first or second author of these published works.

[illegible]
30 September 2009
Jin Yu
30 September 2009

# Acknowledgements

First and foremost, I would like to thank my primary supervisor A. Prof. S. V. N. Vishwanathan for his continuous guidance, support, and encouragement throughout my graduate study. His perpetual energy and enthusiasm for research was always a source of inspiration to me, and I am especially grateful to him for keeping up his supervision after leaving for Purdue University in the United States. I am grateful to my former supervisor, Dr. Nicol N. Schraudolph, who introduced me to the field of optimization, and offered a great deal of help in sharpening my research skills. My sincere appreciation also goes to my panel chair Dr. Jochen Trumpf and my advisor Dr. Knut Hüper for their invaluable advice and support.

I would like to thank Dr. Simon Günter for many constructive discussions and assistance with implementation issues and to Dr. Peter Sunehag for mathematical assistance and proofreading many chapters of this thesis. My appreciation is extended to A. Prof. Jian Zhang for fruitful discussions and collaborative work during my stay at Purdue University.

I was fortunate to work with a group of great researchers at the SML lab, who were so incredibly helpful and supportive. Thanks to each and every one of them. In particular, I am grateful to Dr. Wray Buntine for his help with travel funds, Dr. Marconi Barbosa for keeping our cluster up and running, and Dr. Christfried Webers for his help with Elefant code and proofreading some chapters of this thesis. It is without doubt that life would have been considerably less pleasant if my fellow PhD students were not so fun and great to be with. Their friendship and moral support are deeply appreciated. Especially, I would like to thank Xinhua Zhang and Choonhui Teo for proofreading my papers and assistance with implementation issues. Special thanks go to my friend Bernard Larkin who called regularly to make sure I was doing well.

I am grateful to NICTA and ANU for the scholarship and travel funds that were generously awarded to me. NICTA is funded by the Australian Government's Backing Australia's Ability and the ICT Center of Excellence program. Thanks also go to CECS administrative staff Michelle Moravec, Debbie Pioch, Marie Katselas, and Sue Van Haeften for their indispensable support over these years.

Last but not least, I am in great debt to all of my family for always supporting me in whatever I choose to do. I am particularly grateful to my husband, Dan, for his unconditional love and support, and for just being part of my life.

# Abstract

This thesis develops new quasi-Newton optimization methods that exploit the well-structured functional form of objective functions often encountered in machine learning, while still maintaining the solid foundation of the standard BFGS quasi-Newton method. In particular, our algorithms are tailored for two categories of machine learning problems: (1) regularized risk minimization problems with convex but nonsmooth objective functions and (2) stochastic convex optimization problems that involve learning from small subsamples (mini-batches) of a potentially very large set of data.

We first extend the classical BFGS quasi-Newton method and its limited-memory variant LBFGS to the optimization of nonsmooth convex problems. This is done in a rigorous fashion by generalizing three components of BFGS to subdifferentials: the local quadratic model, the identification of a descent direction, and the Wolfe line search conditions. We prove that under some technical conditions, the resulting subBFGS algorithm is globally convergent in objective function value. We apply the limited-memory variant of subBFGS (subLBFGS) to $L_2$-regularized risk minimization with the binary hinge loss. To extend our algorithms to the multiclass and multilabel settings, we develop a new, efficient, exact line search algorithm. We prove its worst-case time complexity bounds, and show that it can also extend a recently developed bundle method to the multiclass and multilabel settings. Moreover, we apply the direction-finding component of our algorithms to $L_1$-regularized risk minimization with the logistic loss. In all these contexts our methods perform comparable to or better than specialized state-of-the-art solvers on a number of publicly available datasets.

This thesis also provides stochastic variants of the BFGS method, in both full and memory-limited forms, for large-scale optimization of convex problems where objective and gradient must be estimated from subsamples of training data. The limited-memory variant of the resulting online BFGS algorithm performs comparable to a well-tuned natural gradient descent but is scalable to very high-dimensional problems. On standard benchmarks in natural language processing it asymptotically outperforms previous stochastic gradient methods for parameter estimation in Conditional Random Fields.

# Contents

Chapter 1

# Introduction

The goal of most machine learning tasks is to estimate the parameters of a model that enable it to generalize from a set of *training* instances so as to predict correct outputs on previously unseen data. For instance, in the example of Figure 1.1, given a set of training points with their coordinates and the corresponding labels (circles and squares), we can estimate the separating boundary (*e.g.*, the solid line) between the two classes of training points, and then use this model to predict the class of an unlabeled point based on its relative position with respect to the boundary. This example shows that the process of learning essentially involves adaptation of a model to a training dataset. Increasingly, this process is translated into optimizing a convex objective function that measures the performance of the model. The resulting convex optimization problems are challenging because they can involve massive datasets, millions of parameters, nonsmooth functions, and streaming inputs. They often violate common assumptions made by conventional optimization methods, such as differentiability (smoothness) of the objective function and computational tractability of function (*resp.* gradient) evaluation. Efficient and scalable optimization methods that are specifically designed for the machine learning context are therefore needed.

Although conventional methods often fall short of our requirements, they still serve as a good starting point for devising new optimization methods for machine learning. Among dominant conventional optimization methods, the BFGS quasi-Newton method and its limited-memory variant (LBFGS) are widely regarded as the workhorses of smooth nonlinear optimization due to their combination of computational efficiency and good asymptotic convergence. We therefore decided to develop analogous quasi-Newton methods that are tailored for machine learning. In particular, this thesis focuses on two categories of machine learning problems: (1) regularized risk minimization problem that is convex but nonsmooth and (2) stochastic optimization problems that involve learning from small subsamples of the training data.

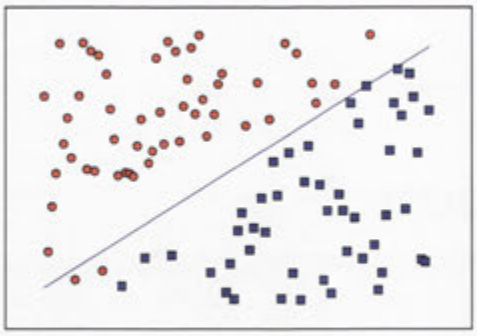


**Figure 1.1:** A simple machine learning problem that involves learning a separating boundary (solid line) between two classes of points (circles and squares).

## 1.1 Motivation

A typical supervised machine learning problem involves a set of training instances that consists of $n$ input feature vectors $\boldsymbol{x}_i$ and their corresponding labels $z_i$. The goal is to build a parametrized model with parameter vector $\boldsymbol{w}$ that can predict correct labels on unseen feature vectors. During the learning process, a domain-specific loss function $l(\boldsymbol{x}_i, z_i, \boldsymbol{w})$ is used to quantify the discrepancy between the true label and the label predicted by the model. The overall performance of the model is then measured by an *empirical risk* $R(\boldsymbol{w})$ that involves the summation of loss terms over the entire set of training data. In order to generalize the model to unseen data, one can employ a *regularizer* $\Omega(\boldsymbol{w})$ that avoids over-fitting the training instances by penalizing complex models. This leads to the following formulation of regularized risk minimization problem with an objective function $J : \mathbb{R}^d \to \mathbb{R}$:

$$J(\boldsymbol{w}) := \lambda\Omega(\boldsymbol{w}) + R(\boldsymbol{w}), \quad \text{where} \tag{1.1}$$
$$R(\boldsymbol{w}) := \frac{1}{n}\sum_{i=1}^{n} l(\boldsymbol{x}_i, z_i, \boldsymbol{w}).$$

The regularization constant $\lambda > 0$ is a free parameter trading off the model complexity and the empirical performance in terms of the average loss on the training data. Typically, the regularizer $\Omega(\boldsymbol{w})$ is easy to compute but the empirical risk $R(\boldsymbol{w})$ is not, due to the presence of a summation over the entire training data. The resulting objective function is convex but not necessarily differentiable everywhere. Minimizing $J$ gives the desired parameter $\boldsymbol{w}^*$. The model parametrized by $\boldsymbol{w}^*$ is then used to predict labels of unseen data. This general framework underlines many problems in machine learning. Here we provide two representative examples: $L_2$-regularized risk minimization with

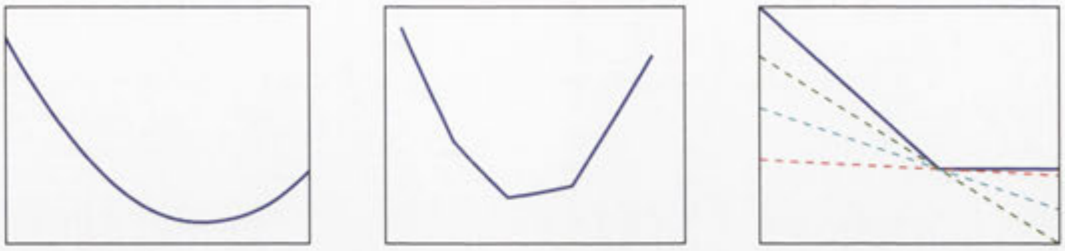


**Figure 1.2:** Left: the objective function of a typical regularized risk minimization problem ( plotted along a direction) is convex, but zooming into the region around the optimum (center) reveals its nonsmooth points. Right: the hinge loss: $l(f) := \max(0, 1 - f)$ (solid line) is nonsmooth; the slope of any line that is tangiantial to $l$ at a point (*e.g.*, dashed lines) is a subgradient.

the hinge loss and $L_1$-regularized risk minimization with the logistic loss.

$L_2$-regularized risk minimization uses a quadratic regularizer $\Omega(\boldsymbol{w}) := \frac{1}{2}\,\boldsymbol{w}^\top\boldsymbol{w}$, where the superscript $\top$ denotes transpose. Binary classification as a typical machine learning task considers the problem of differentiating between two classes of objects (*cf.* Figure 1.1). A common loss function for binary classification is the (binary) hinge loss

$$l(\boldsymbol{x}, z, \boldsymbol{w}) \;:=\; \max(0, 1 - z\,\boldsymbol{w}^\top\boldsymbol{x}), \tag{1.2}$$

which measures the discrepancy between the correct label $z \in \{\pm 1\}$ and the prediction given by $\operatorname{sign}(\boldsymbol{w}^\top\boldsymbol{x})$. Obviously, the hinge loss is convex but nonsmooth at points where $z\,\boldsymbol{w}^\top\boldsymbol{x} = 1$ (Figure 1.2, right). Figure 1.2 (left) provides a one-dimensional view of the resulting objective function $J$: the zoomed-in figure (Figure 1.2, center) shows that $J$ is nonsmooth. This means that many standard optimizers such as conjugate gradient (Shewchuk, 1994) and quasi-Newton methods (Nocedal and Wright, 1999) are not suitable here because they are not able to deal with nonsmooth functions.

Another popular approach to binary classification is to build a probabilistic classifier that models a conditional probability of a label $z$ given a feature vector $\boldsymbol{x}$, and outputs the most likely label as its prediction on a given $\boldsymbol{x}$. A widely used probabilistic model for this task is the log-linear model $P(z|\boldsymbol{x};\boldsymbol{w}) := 1/(1 + e^{-z\boldsymbol{w}^\top\boldsymbol{x}})$, parametrized by $\boldsymbol{w}$. During the process of learning, the log-linear model is adapted to assign a higher probability to the true label. This is achieved by minimizing the logistic loss $-\ln P(z|\boldsymbol{x};\boldsymbol{w}) = \ln(1+e^{-z\boldsymbol{w}^\top\boldsymbol{x}})$, *i.e.*, the negative log-likelihood of the log-linear model. The $L_1$ regularizer $\|\boldsymbol{w}\|_1 := \sum_{i=1}^{d} |w_i|$ is commonly used to enforce sparsity in the solution $\boldsymbol{w}$ of dimensionality $d$, leading to an objective function of the form

$$J(\boldsymbol{w}) := \lambda\|\boldsymbol{w}\|_1 + \frac{1}{n}\sum_{i=1}^{n}\ln(1 + e^{-z_i\boldsymbol{w}^\top\boldsymbol{x}_i}), \tag{1.3}$$

where the loss is smooth, but the regularizer is nonsmooth at points where $\boldsymbol{w}$ has zero elements. Again, standard optimizers for smooth optimization can not be applied here.

The above examples characterize many machine learning problems which are challenging to solve in general but are often endowed with very special structure. For instance, the nonsmoothness of the example problems arises both due to the presence of *piecewise linear* terms in their objective functions. As will be shown in later chapters, leveraging this special structure of the objective function can greatly reduce the complexity of solving the problem. General purpose optimizers like the widely used bundle methods (Hiriart-Urruty and Lemaréchal, 1993), however, do not take any advantage of the special structure inherent to a specific problem. This constrains their potential to be applied to a wider range of machine learning problems. This thesis sets out to develop new optimization methods that exploit the well-structured functional form of objective functions encountered in machine learning. In particular, we are interested in regularized risk minimization problems that are convex but nonsmooth.

## 1.2 BFGS Quasi-Newton Methods

The Broyden-Fletcher-Goldfarb-Shanno (BFGS) quasi-Newton method (Dennis and Moré, 1977) was invented independently by Broyden, Flecher, Goldfarb, and Shanno in the early seventies. It revolutionized smooth nonlinear optimization, and has dominated it to this date, due to its superior practical performance. Given a smooth objective function $J : \mathbb{R}^d \to \mathbb{R}$ and a current iterate $\boldsymbol{w}_t \in \mathbb{R}^d$, BFGS forms a local quadratic model of $J$:

$$Q_t(\boldsymbol{p}) \ := \ J(\boldsymbol{w}_t) + \tfrac{1}{2}\boldsymbol{p}^\top\boldsymbol{B}_t^{-1}\boldsymbol{p} + \nabla J(\boldsymbol{w}_t)^\top\boldsymbol{p}, \tag{1.4}$$

where $\boldsymbol{B}_t$ is a symmetric positive definite approximation to the inverse Hessian $\boldsymbol{H}_t^{-1}$ of $J$, and $\nabla J$ denotes the gradient. Minimizing $Q_t(\boldsymbol{p})$ gives the quasi-Newton direction

$$\boldsymbol{p}_t := -\boldsymbol{B}_t\nabla J(\boldsymbol{w}_t), \tag{1.5}$$

which is used for the parameter update:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \eta_t\boldsymbol{p}_t, \tag{1.6}$$

where the step size $\eta_t > 0$ controls how far to move in the given direction $\boldsymbol{p}_t$. $\eta_t$ is commonly determined by a *line search* such that a sufficient decrease in the objective value is achieved. In other words, the goal of a line search is to decrease the one-dimensional function $\Phi$ of the step size $\eta$:

$$\Phi(\eta) := J(\boldsymbol{w}_t + \eta \boldsymbol{p}_t). \tag{1.7}$$

There are two kinds of line search strategies: Exact line search finds the optimal step size by minimizing $\Phi(\eta)$ but is only feasible when the functional form of $\Phi$ is known and amenable to efficient minimization (*e.g.*, explicit solution). Inexact line searches, on the other hand, only require the access to function and gradient values. They only seek to find an "appropriate" step size (quantified in terms of line search conditions, *e.g.*, Wolfe conditions (2.7) and (2.8)).

After each parameter update, the $\boldsymbol{B}_t$ matrix is modified via the incremental update

$$\boldsymbol{B}_{t+1} = (\boldsymbol{I} - \rho_t \boldsymbol{s}_t \boldsymbol{y}_t^\top)\boldsymbol{B}_t(\boldsymbol{I} - \rho_t \boldsymbol{y}_t \boldsymbol{s}_t^\top) + \rho_t \boldsymbol{s}_t \boldsymbol{s}_t^\top, \tag{1.8}$$

where

$$\boldsymbol{s}_t := \boldsymbol{w}_{t+1} - \boldsymbol{w}_t \text{ and } \boldsymbol{y}_t := \nabla J(\boldsymbol{w}_{t+1}) - \nabla J(\boldsymbol{w}_t) \tag{1.9}$$

denote the most recent steps along the optimization trajectory in parameter and gradient space, respectively, and $\rho_t := (\boldsymbol{y}_t^\top \boldsymbol{s}_t)^{-1}$. The process repeats until the norm of the gradient falls below a pre-specified threshold. Note that replacing $\boldsymbol{B}_t$ in (1.5) with the inverse Hessian $\boldsymbol{H}_t^{-1}$ recovers the familiar Newton direction (Nocedal and Wright, 1999). Although Newton's method has faster rate of convergence (in terms of iteration numbers) than BFGS (quadratic *vs.* super-linear rate), its $O(d^3)$ cost of inverting $\boldsymbol{H}_t$ can be prohibitive. Unlike Newton's method, BFGS uses past parameter and gradient displacements (1.9) to build an approximation $\boldsymbol{B}_t$ to the inverse Hessian, reducing the cost per iteration to $O(d^2)$.

Liu and Nocedal (1989) proposed a scalable variant of the BFGS method, called limited-memory BFGS (LBFGS), for solving high-dimensional problems where the $O(d^2)$ cost of storing and updating $\boldsymbol{B}_t$ would be prohibitive. LBFGS does not maintain an approximation matrix to the inverse Hessian. Instead, it approximates the quasi-Newton direction (1.5) directly via a recursive procedure (Algorithm 2.3). LBFGS has become the algorithm of choice for high-dimensional smooth nonlinear problems, due to its scalability and good asymptotic convergence inherited from BFGS.

## 1.3 Quasi-Newton Methods for Nonsmooth Optimization

There have been some attempts to apply (L)BFGS directly to nonsmooth optimization problems in the hope that these dominant algorithms for smooth optimization would also perform well on nonsmooth functions that are convex and differentiable almost everywhere; otherwise a subgradient (generalized gradient for nonsmooth functions) exists. Indeed, it has been noted that in cases where BFGS (*resp.* LBFGS) does not encounter any nonsmooth point, it often converges to the optimum (Lemarechal, 1982; Lewis and Overton, 2008a). However, Lewis and Overton (2008b), Lukšan and Vlček (1999) and Haarala (2004) also report catastrophic failures of (L)BFGS on nonsmooth functions. This has motivated various modifications to BFGS and LBFGS to facilitate their use on nonsmooth problems. In what follows, we briefly discuss these modifications, before introducing our quasi-Newton approach to nonsmooth optimization.

### 1.3.1 Existing Approaches

Various modifications to BFGS (*resp.* LBFGS) (Haarala, 2004; Lukšan and Vlček, 1999; Rauf and Fukushima, 1996) have been proposed in order to ensure its convergence on nonsmooth problems. A common feature of these modifications is that they require repeated evaluation of function (and subgradient) around nonsmooth points (Haarala, 2004; Lukšan and Vlček, 1999), or in some cases also around smooth points (Rauf and Fukushima, 1996) so as to build a faithful local model of the objective function. In most machine learning problems, *e.g.*, our targeted regularized risk minimization problems, the objective function (and hence its subgradient) sums contributions from every instance in a set of training data. When learning on massive datasets with millions of training instances, function (*resp.* subgradient) evaluation is computationally very expensive. Therefore, existing extensions of (L)BFGS to nonsmooth problems are not suitable for our problems. In contrast to these approaches, we build a local model of the objective function from subgradients evaluated only at a single nonsmooth point. This can be done very efficiently because the special structure present in our problems allows for the exact evaluation of *all* subgradients at a nonsmooth point.

Another possible way to bypass the complications caused by lack of smoothness of an objective function is to work on a smooth approximation instead (Nemirovski, 2005; Nesterov, 2005). Although this approach has met with some success in recent years, it is unclear how to build a smooth approximation in general. Furthermore, smooth loss functions do not preserve sparsity in the solution of a machine learning problem, which often leads to good generalization performance on unseen data. This thesis therefore focuses on the underlying optimization problems, and not on the modeling issues such as the choice of loss function. In what follows we provide a high-level discussion of our approach to extend (L)BFGS to nonsmooth objective functions.

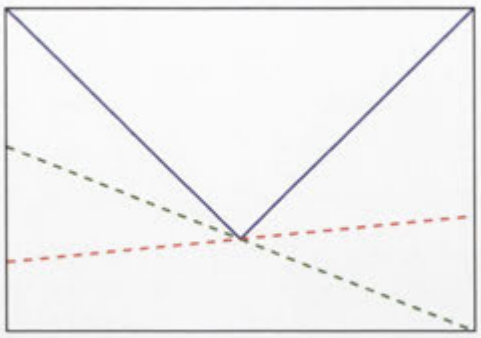


**Figure 1.3:** The gradients of the dashed lines give two subgradients of the function (solid line) at its nonsmooth point (the optimum). Adjusting the parameter in the negative direction of any of the two subgradients takes us out of the optimality.

### 1.3.2 Our Approach

The fundamental reason for the deficiency of (L)BFGS on a nonsmooth function is that it may not be able to determine a *descent* (downhill) direction to decrease the objective function value at a non-differentiable point. Although a convex function might not be differentiable everywhere, a subgradient always exists (Hiriart-Urruty and Lemaréchal, 1993). Let $\boldsymbol{w}$ be a point where a convex function $J$ is finite. Then a subgradient is the normal vector to any tangential supporting hyperplane of $J$ at $\boldsymbol{w}$. Formally, $\boldsymbol{g}$ is called a subgradient of $J$ at $\boldsymbol{w}$ if and only if (Hiriart-Urruty and Lemaréchal, 1993, Definition VI.1.2.1)

$$(\forall \boldsymbol{w}')\;\; J(\boldsymbol{w}') \;\geq\; J(\boldsymbol{w}) + (\boldsymbol{w}' - \boldsymbol{w})^\top \boldsymbol{g}. \tag{1.10}$$

The set of all subgradients at a point is called the subdifferential, and is denoted $\partial J(\boldsymbol{w})$. If this set is not empty, then $J$ is said to be *subdifferentiable at* $\boldsymbol{w}$. If it contains exactly one element, *i.e.*, $\partial J(\boldsymbol{w}) = \{\nabla J(\boldsymbol{w})\}$, then $J$ is *differentiable* at $\boldsymbol{w}$. Figure 1.2 (right) provides the geometric illustration of (1.10).

Recall that BFGS assumes the objective function $J$ is differentiable everywhere so that at the current iterate $\boldsymbol{w}_t$ it can construct a local quadratic model (1.4) of $J(\boldsymbol{w}_t)$. For a nonsmooth objective function, such a model becomes ambiguous at nonsmooth points: replacing the gradient $\nabla J(\boldsymbol{w}_t)$ in (1.4) with different subgradients yields different models. To resolve the ambiguity, we could simply replace the gradient $\nabla J(\boldsymbol{w}_t)$ in (1.4) with an arbitrary subgradient $\boldsymbol{g}_t \in \partial J(\boldsymbol{w}_t)$, and minimize this model to obtain the quasi-Newton direction $\boldsymbol{p}_t := -\boldsymbol{B}_t\boldsymbol{g}_t$, which, however, is not necessarily a direction of descent. This is essentially caused by the fact that the negative direction of a subgradient need not be a descent direction, as Figure 1.3 illustrates. Formally, a direction

$\boldsymbol{p}_t$ is a descent direction at a point $\boldsymbol{w}_t$ if and only if

$$\sup_{\boldsymbol{g} \in \partial J(\boldsymbol{w}_t)} \boldsymbol{g}^\top \boldsymbol{p}_t < 0. \tag{1.11}$$

Since $\boldsymbol{p}_t$ may not fulfill this condition, it may not be possible for a line search to find a valid step size $\eta_t > 0$. To fix this fundamental modeling problem, we propose a new model that takes the supremum over all possible quadratic models generated from different subgradients taking the place of the gradient in the BFGS quadratic model (1.4). This corresponds to replacing the last term in (1.4) with $\sup_{\boldsymbol{g} \in \partial J(\boldsymbol{w}_t)} \boldsymbol{g}^\top \boldsymbol{p}_t$.

Having constructed a local model of $J$, we can minimize it to obtain a direction of descent. Minimizing our new model is itself a challenging task due to the presence of the supremum over the entire set of subgradients. However, since the nonsmoothness in our problem stems only from *piecewise linear* terms in the objective function, the subdifferential is very well structured. Specifically, $\partial J(\boldsymbol{w}_t)$ is a convex and compact polyhedron characterised as the convex hull of its extreme points. For instance, all subgradients of the piecewise linear function shown in Figure 1.2 (right) form the interval between the gradients of its left and right linear segments. The fact that the supremum over a polyhedral set can only be attained at an extreme point (Bertsekas, 1999, Proposition B.21c) allows us to easily compute $\sup_{\boldsymbol{g} \in \partial J(\boldsymbol{w}_t)} \boldsymbol{g}^\top \boldsymbol{p}_t$. Based on this observation, we are able to develop an efficient iterative procedure that is guaranteed to produce a quasi-Newton direction that satisfies the descent condition (1.11).

Given a descent direction, we need to find a step size that reduces the objective function value in this direction, *i.e.*, reduces the value of the one-dimensional function $\Phi$ (1.7). Since $\Phi$ is simply the objective function $J$ restricted to a line, the structure of $J$ is preserved in $\Phi$, *e.g.*, if $J$ is piecewise quadratic, then so is $\Phi$. Using this knowledge, we can not only generalize standard inexact line searches to the nonsmooth setting, but also develop efficient exact line searches that take into account the structured functional form of $\Phi$.

## 1.4 Stochastic Quasi-Newton Methods

As we have already seen (*e.g.*, in (1.1)), machine learning poses data-driven optimization problems in which the objective function involves the summation of loss terms over a set of data to be modeled. Classical optimization techniques must compute this sum in its entirety for each evaluation of the objective function, respectively its gradient. As available datasets grow ever larger, such "batch" (deterministic) optimizers therefore become increasingly inefficient. They are also ill-suited for the online (incremental) setting, where partial data must be modeled as it arrives.

Stochastic (online) gradient methods, by contrast, work with function and gradient estimates obtained from small subsamples (mini-batches) of the data. The stochastic approximation of the regularized risk (1.1), for instance, takes the form

$$J(\boldsymbol{w}, \mathcal{X}) := \lambda\Omega(\boldsymbol{w}) + \frac{1}{b} \sum_{(\boldsymbol{x}_i, z_i) \in \mathcal{X}} l(\boldsymbol{x}_i, z_i, \boldsymbol{w}) \tag{1.12}$$

where $\mathcal{X}$ is a *mini-batch* of $b$ training instances $(\boldsymbol{x}_i, z_i)$, *i.e.*, pairs of feature vectors and their corresponding correct labels drawn from the set of training data. It has been noted (Bottou, 1998) that stochastic methods are generally robust to the nonsmoothness of the objective function. Intuitively, this is because we can always take another batch of data to avoid landing on the same nonsmooth points.

Since the batch size $b$ in (1.12) is usually much less than the size of the training set, function (and hence gradient) evaluation in the stochastic setting is cheap. This can greatly reduce computational requirements: on large, redundant datasets, simple stochastic gradient descent:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta_t \nabla_{\boldsymbol{w}} J(\boldsymbol{w}_t, \mathcal{X}_t) \tag{1.13}$$

routinely outperforms sophisticated second-order batch methods, *e.g.*, LBFGS, by orders of magnitude (Bottou, 2009; Vishwanathan et al., 2006), in spite of the slow convergence of first-order gradient descent. In the stochastic setting the step size $\eta_t$ is commonly decayed over iterations, *e.g.*, by a decay schedule

$$\eta_t = \frac{\tau}{\tau + t}\,\eta_0\,, \tag{1.14}$$

where $\eta_0, \tau > 0$ are tuning parameters. Schraudolph (1999, 2002) further accelerates stochastic gradient descent through online adaptation of a step size vector.

Attempts to develop more advanced stochastic methods are hampered by the fact that core tools of conventional gradient-based optimization, such as line searches and Krylov subspaces, are not amenable to stochastic approximation (Schraudolph and Graepel, 2003): online implementations of conjugate gradient methods (Møller, 1993; Schraudolph and Graepel, 2003), for instance, have proven largely ineffective.

Natural gradient descent (NG, Amari et al., 1998) is an online second-order learning algorithm that works by incrementally maintaining an approximation to the inverse of

$$\mathbb{E}_{\mathcal{X}}[\nabla_{\boldsymbol{w}} J(\boldsymbol{w}_t, \mathcal{X}) \nabla_{\boldsymbol{w}} J(\boldsymbol{w}_t, \mathcal{X})^\top] \tag{1.15}$$

(the covariance matrix of the stochastic gradient), which is then used to scale the parameter update. While quite effective, NG does not model the curvature (Hessian) of the objective function, and requires $O(d^2)$ space and time per iteration to optimize a system with $d$ parameters.

We overcome these limitations by systematically modifying BFGS so as to make it amenable to stochastic approximation of gradients. The changes required to get BFGS to work well with stochastic approximation fall into three aspects: making do without a line search, modifying the update of BFGS' inverse Hessian approximation (1.8), and taking consistent gradient measurements for the calculation of $\boldsymbol{y}_t$ in (1.9). Moreover, by applying analogous modifications to LBFGS we are able to obtain a stochastic LBFGS method.

## 1.5 Thesis Contributions

The major contributions of this thesis are:

1. We systematically extend the classical quasi-Newton framework for smooth nonlinear optimization to nonsmooth objectives. The resulting quasi-Newton algorithms are amenable to subgradients and proven to converge to the optimal objective value. In addition, our algorithms are able to take advantage of the polyhedral structure present in the subdifferential of nonsmooth objective functions often encountered in machine learning. This allows our methods to perform competitively when benchmarked against state-of-the-art machine learning solvers on a range of machine learning problems.

2. We develop new exact line search methods specialized for $L_2$-regularized risk minimization with the hinge loss (1.2) and its generalizations to the more challenging multiclass and multilabel classification problems. In the multiclass setting the class label $z$ can take any integer value instead of being restricted to the set $\{\pm 1\}$, while in the multilabel setting multiple labels can be assigned to one feature vector, *i.e.*, $z$ becomes a set. By exploiting the piecewise linear structure in this class of convex but nonsmooth classification problems, our exact line search methods efficiently find the optimal step size that minimizes the objective function in a given search direction. These line search methods can be used as black-box procedures to accelerate the convergence of any adaptive classifier whose parameter update takes the form of (1.6).

3. Stochastic variants of BFGS and LBFGS are also developed in this thesis. To the best of our knowledge, this is the first successful extension of the standard quasi-Newton methods to the stochastic setting. Stochastic LBFGS, in particular, is significant as the first stochastic gradient algorithm which combines the desirable properties of quasi-Newton methods with good scaling to both large datasets and large models (*i.e.*, with many parameters).

## 1.6 Outline

The rest of this thesis is organized as follows:

- **Chapter 2:**
  This chapter provides an overview of the standard BFGS quasi-Newton method and its limited-memory variant (LBFGS). In particular, we describe in detail the three building blocks of the BFGS method: (1) the local quadratic model, (2) the line search method, and (3) the BFGS inverse Hessian approximation.

- **Chapter 3:**
  We extend standard (L)BFGS to nonsmooth convex optimization. This is done in a rigorous fashion by extending key components of BFGS to subdifferentials. We then demonstrate the use of the resulting subBFGS (*resp.* subLBFGS) algorithm for regularized risk minimization with various hinge losses for binary, multiclass, and multilabel classification tasks.

- **Chapter 4:**
  An extensive empirical evaluation of subLBFGS is carried out in this chapter. We compare the performance of subLBFGS with specialized state-of-the-art machine learning solvers on $L_2$-regularized risk minimization with various hinge losses. We also apply the direction-finding component of our algorithm to $L_1$-regularized risk minimization with the logistic loss.

- **Chapter 5:**
  We develop stochastic variants of BFGS in both full and memory-limited forms. The resulting algorithms demonstrate competitive performance in comparison to previous stochastic approaches.

- **Chapter 6:**
  The online LBFGS method is applied to parameter estimation in Conditional Random Fields with over $10^5$ parameters, as used in natural language processing.

- **Chapter 7:**
  We conclude with a summary of the thesis and ideas for future work.

## 1.7 Notation

This section describes notational conventions used throughout this thesis. Scalars are denoted by non-bold letters, *e.g.*, $x$. Vectors are denoted by lowercase boldface letters, *e.g.*, $\boldsymbol{x}$, and matrices by capital boldface letters, *e.g.*, $\boldsymbol{A}$. We use boldface numbers to denote a vector of that number: $\mathbf{1}$ denotes a vector of all ones and $\mathbf{0}$ a vector of all zeros. Calligraphic letters refer to sets, but we denote the set of natural numbers $\mathbb{N}$, the set of integers $\mathbb{Z}$, the set of real numbers $\mathbb{R}$ and the set of $d$-dimensional real vectors $\mathbb{R}^d$.

Vectors are viewed as column vectors. A superscript $\top$ denotes the transpose of a vector or matrix. For a vector $\boldsymbol{x} \in \mathbb{R}^d$, $\boldsymbol{x}^\top$ is therefore a $d$-dimensional row vector. The inner product of two vectors $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$ is written as $\boldsymbol{x}^\top \boldsymbol{y}$. To denote the $i^{\text{th}}$ element of a vector $\boldsymbol{x}$, we use the notation $x_i$; the entry of a matrix $\boldsymbol{A}$ at the $i^{\text{th}}$ row and the $j^{\text{th}}$ column is denoted by $\boldsymbol{A}_{ij}$. $[\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_n]$ denotes a matrix with columns $\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_n$. For an invertible matrix $\boldsymbol{A}$, $\boldsymbol{A}^{-1}$ denotes its inverse. We use $l \preceq \boldsymbol{A} \preceq u$ to express that all eigenvalues of $\boldsymbol{A}$ lie between $l$ and $u$. We use $\|\cdot\|$ as a shorthand for the $L_2$ (Euclidean) norm, *i.e.*, $\|\boldsymbol{x}\| := \sqrt{\boldsymbol{x}^\top \boldsymbol{x}}$; for a matrix, $\|\cdot\|$ denotes the matrix norm induced by the $L_2$ vector norm, *i.e.*, $\|\boldsymbol{A}\| := \max_{\boldsymbol{x} \neq \boldsymbol{0}} \frac{\|\boldsymbol{A}\boldsymbol{x}\|}{\|\boldsymbol{x}\|}$. We use $\|\cdot\|_1$ to denote the $L_1$ vector norm, *i.e.*, $\|\boldsymbol{x}\|_1 := \sum_i |\boldsymbol{x}_i|$.

If $J$ is a function, we use the notation $J : \mathcal{A} \to \mathcal{B}$ to indicate that $J$ is defined on a set $\mathcal{A}$, and takes values from a set $\mathcal{B}$. Following the notational conventions in convex analysis, we use $\nabla J$ to denote the gradient of a differentiable function $J$; if $J$ is non-differentiable, we use $\partial J$ to denote its subdifferential. The expectation of a function $J(\boldsymbol{x}, \boldsymbol{y})$ with respect to a random variable $\boldsymbol{x}$ is denoted by $\mathbb{E}_{\boldsymbol{x}}[J(\boldsymbol{x}, \boldsymbol{y})]$. When it is clear which random variable an expectation is taken over, we omit the subscript, *e.g.*, $\mathbb{E}(\boldsymbol{x}\boldsymbol{x}^\top)$.

The "big O" notation $O(\cdot)$ is used in this thesis to characterize the size of a quantity or the computational complexity of an algorithm. We write $y = O(x)$ if and only if there exists a constant $c > 0$ such that for any value of $x \in \mathbb{R}$, $|y| \leq cx$. For vectors and matrices, we use $O(\cdot)$ to quantify their Euclidean norms. For instance, for a vector $\boldsymbol{y} \in \mathbb{R}^d$, we write $\boldsymbol{y} = O(x)$ if and only if there exists a constant $c > 0$ such that $(\forall x \in \mathbb{R})\ \|\boldsymbol{y}\| \leq cx$.

Subscript $t$ is reserved as an iteration (time) index: $\boldsymbol{x}_t$ means the value of $\boldsymbol{x}$ at iteration $t$. When there is an iterative sub-procedure within a main iteration, we use the superscript $(i)$ to index the sub-iteration: $\boldsymbol{x}_t^{(i)}$ denotes the value of $\boldsymbol{x}_t$ after $i$ iterations of the sub-procedure.

Chapter 2

# Classical Quasi-Newton Methods

In this chapter we review the Broyden-Fletcher-Goldfarb-Shanno (BFGS) quasi-Newton method (Dennis and Moré, 1977) in both its full (Section 2.1) and memory-limited forms (Section 2.2). Throughout this chapter we assume the objective function $J : \mathbb{R}^d \rightarrow \mathbb{R}$ is convex, deterministic, and continuously differentiable everywhere. The new quasi-Newton methods developed in Chapters 3 and 5 will relax these constraints, while maintaining the solid foundation of this classical optimization technique.

## 2.1 The BFGS Quasi-Newton Method

The BFGS quasi-Newton algorithm (Dennis and Moré, 1977; Fletcher, 1989; Nocedal and Wright, 1999) was invented independently by Broyden, Flecher, Goldfarb, and Shanno in the early seventies. It is by far the most successful quasi-Newton method for unconstrained smooth nonlinear optimization due to its combination of computational efficiency and good asymptotic convergence. In what follows, we review this algorithm (Algorithm 2.1), focusing on its three key components, namely, the local quadratic model, the line search method, and the inverse Hessian approximation.

### 2.1.1 Local Quadratic Model

Given a continuously differentiable objective function: $J : \mathbb{R}^d \rightarrow \mathbb{R}$ and a current iterate $\boldsymbol{w}_t \in \mathbb{R}^d$, BFGS forms a local quadratic model of $J$:

$$Q_t(\boldsymbol{p}) \; := \; J(\boldsymbol{w}_t) + \tfrac{1}{2}\boldsymbol{p}^\top \boldsymbol{B}_t^{-1}\boldsymbol{p} + \nabla J(\boldsymbol{w}_t)^\top \boldsymbol{p}, \tag{2.1}$$

where $\boldsymbol{B}_t$ is a symmetric positive definite estimate of the inverse Hessian of $J$ (assuming that $J$ is twice-differentiable), and $\nabla J$ denotes the gradient. The quadratic model (2.1) can be seen as an approximation to a truncated second-order Taylor expansion of $J$

**Algorithm 2.1** CLASSICAL BFGS METHOD

```
1: Initialize: $t := 0$, $\boldsymbol{B}_0 = \boldsymbol{I}$, and $\boldsymbol{w}_0$
2: Set: convergence tolerance $\epsilon > 0$
3: while $\|\nabla J(\boldsymbol{w}_t)\| > \epsilon$ do
4:     $\boldsymbol{p}_t = -\boldsymbol{B}_t \nabla J(\boldsymbol{w}_t)$
5:     Find $\eta_t$ that obeys (2.7) and (2.8)
6:     $\boldsymbol{s}_t = \eta_t \boldsymbol{p}_t$
7:     $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \boldsymbol{s}_t$
8:     $\boldsymbol{y}_t = \nabla J(\boldsymbol{w}_{t+1}) - \nabla J(\boldsymbol{w}_t)$
9:     if $t = 0$ then
10:        $\boldsymbol{B}_t := \frac{\boldsymbol{s}_t^\top \boldsymbol{y}_t}{\boldsymbol{y}_t^\top \boldsymbol{y}_t} \boldsymbol{I}$
11:    end if
12:    $\rho_t = (\boldsymbol{s}_t^\top \boldsymbol{y}_t)^{-1}$
13:    $\boldsymbol{B}_{t+1} = (\boldsymbol{I} - \rho_t \boldsymbol{s}_t \boldsymbol{y}_t^\top) \boldsymbol{B}_t (\boldsymbol{I} - \rho_t \boldsymbol{y}_t \boldsymbol{s}_t^\top) + \rho_t \boldsymbol{s}_t \boldsymbol{s}_t^\top$
14:    $t := t + 1$
15: end while
```

around $\boldsymbol{w}_t$:

$$Q_t(\boldsymbol{p}) \approx J(\boldsymbol{w}_t) + \tfrac{1}{2}\boldsymbol{p}^\top \boldsymbol{H}_t \boldsymbol{p} + \nabla J(\boldsymbol{w}_t)^\top \boldsymbol{p} \approx J(\boldsymbol{w}_t + \boldsymbol{p}), \tag{2.2}$$

where $\boldsymbol{H}_t$ is the Hessian (second-order derivative of $J$) at $\boldsymbol{w}_t$. Taking the derivative of (2.1) and setting it to zero give the so-called quasi-Newton direction:

$$\boldsymbol{p}_t := -\boldsymbol{B}_t \nabla J(\boldsymbol{w}_t), \tag{2.3}$$

which is always a direction of descent, *i.e.*, along $\boldsymbol{p}_t$ the objective function value can be decreased. Formally, a direction $\boldsymbol{p} \in \mathbb{R}^d$ is a descent direction at an iterate $\boldsymbol{w}$ if and only if it satisfies $\nabla J(\boldsymbol{w})^\top \boldsymbol{p} < 0$. This is true for the quasi-Newton direction (2.3) because

$$\nabla J(\boldsymbol{w}_t)^\top \boldsymbol{p}_t = -\nabla J(\boldsymbol{w}_t)^\top \boldsymbol{B}_t \nabla J(\boldsymbol{w}_t) < 0 \tag{2.4}$$

holds due to the positivity of $\boldsymbol{B}_t$.

Given the quasi-Newton direction $\boldsymbol{p}_t$, BFGS adjusts the parameters by taking a step along this descent direction (Lines 6–7 of Algorithm 2.1):

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \eta_t \boldsymbol{p}_t, \tag{2.5}$$

where the step size $\eta_t > 0$ is normally determined by a line search procedure (Line 5 of Algorithm 2.1, see also Section 2.1.2) that enforces technical conditions to ensure global convergence to the optimum of $J$. The parameter update (2.5) is carried out

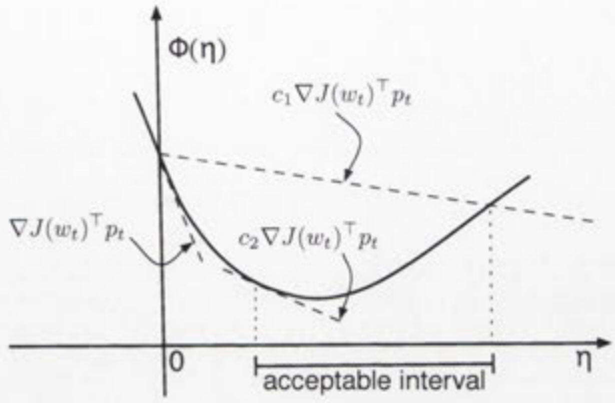


**Figure 2.1**: Geometric illustration of the Wolfe conditions (2.7) and (2.8).

iteratively until the norm of the gradient drops below a pre-specified small tolerance, indicating that the solution is within a close neighbourhood of the optimum.

### 2.1.2 Line Search

Given the current iterate $\boldsymbol{w}_t$ and a descent search direction $\boldsymbol{p}_t$, the task of a line search is to determine how far to move along the ray: $(\boldsymbol{w}_t + \eta\boldsymbol{p}_t)$ with $\eta > 0$ to reduce the value of the objective function, *i.e.*, reducing the value of the one-dimensional function

$$\Phi(\eta) := J(\boldsymbol{w}_t + \eta\boldsymbol{p}_t). \tag{2.6}$$

Exact line search finds the optimal step size by minimizing $\Phi(\eta)$ but is only feasible when the exact functional form of the objective function (and hence $\Phi$) is known and amenable to efficient minimization. This is, in general, not possible. However, we can often do it in machine learning because objective functions (*resp.* $\Phi$) of most machine learning problems are explicitly given.

Inexact line searches only require the access to function and gradient values. They only seek to minimize (2.6) approximately by enforcing conditions designed to ensure convergence. As implemented here (Algorithm 2.1), BFGS uses an inexact line search that obeys the Wolfe conditions (Wolfe, 1969):

$$J(\boldsymbol{w}_{t+1}) \leq J(\boldsymbol{w}_t) + c_1\eta_t\nabla J(\boldsymbol{w}_t)^\top\boldsymbol{p}_t \qquad \text{(sufficient decrease)} \tag{2.7}$$

$$\text{and} \quad \nabla J(\boldsymbol{w}_{t+1})^\top\boldsymbol{p}_t \geq c_2\nabla J(\boldsymbol{w}_t)^\top\boldsymbol{p}_t, \qquad \text{(curvature)} \tag{2.8}$$

with $0 < c_1 < c_2 < 1$. Typical values for $c_1$ and $c_2$ are $10^{-4}$ and 0.9, respectively. The Wolfe conditions facilitate global convergence by guaranteeing a sufficient decrease in the value of the objective function and excluding pathologically small step sizes via (2.7) and (2.8), respectively (Nocedal and Wright, 1999, Thorem 3.2 and 8.5). Figure 2.1 illustrates this geometrically.

A natural question to ask is whether the optimal step size $\eta^*$ obtained by exact line search satisfies the Wolfe conditions. The answer is no because depending on the choice of $c_1$, $\eta^*$ may violate the sufficient decrease condition (2.7). For instance, for the function plotted in Figure 2.1, we can increase the value of $c_1$ such that the acceptable interval for the step size excludes $\eta^*$. To prevent this from happening, in practical implementations $c_1$ is often set to a small value, *e.g.*, $10^{-4}$. On the other hand, the curvature condition (2.8) is always satisfied by $\eta^*$:

$$\nabla J(\boldsymbol{w}_t + \eta^* \boldsymbol{p}_t)^\top \boldsymbol{p}_t \;=\; 0 \;>\; J(\boldsymbol{w}_t)^\top \boldsymbol{p}_t \tag{2.9}$$

because $\boldsymbol{p}_t$ is a descent direction (2.4) and the gradient of $\Phi$:

$$\nabla \Phi(\eta) = \nabla J(\boldsymbol{w}_t + \eta \boldsymbol{p}_t)^\top \boldsymbol{p}_t \tag{2.10}$$

vanishes at $\eta^*$.

The most commonly used inexact line search procedure is a backtracking line search that obeys the Wolfe conditions. It tries candidate steps of the form $\eta_0 \beta^k$ for $k = 0, 1, 2, \ldots$ until (2.7) and (2.8) are satisfied, where $0 < \beta < 1$ is a decay factor, and $\eta_0 > 0$ an initial step size that satisfies the curvature condition (2.8), *i.e.*, $\eta_0$ must not be less than the minimal acceptable step size as illustrated in Figure 2.1. Bertsekas (1999, Proposition 1.2.1) shows that a backtracking line search that obeys the sufficient descent direction (2.7) can already guarantee global convergence to the optimum of a convex and smooth objective function, provided that all search directions supplied to the backtracking line search are descent directions.

## 2.1.3 Inverse Hessian Approximation

The BFGS' symmetric positive-definite approximation $\boldsymbol{B}_t$ to the inverse Hessian (curvature) plays a key role in forming the quasi-Newton direction (2.3). It is maintained incrementally alongside the parameter update (2.5): as BFGS moves to the new iterate $\boldsymbol{w}_{t+1}$, it adjusts its quadratic model to

$$Q_{t+1}(\boldsymbol{p}) \;:=\; J(\boldsymbol{w}_{t+1}) + \tfrac{1}{2}\boldsymbol{p}^\top \boldsymbol{B}_{t+1}^{-1} \boldsymbol{p} + \nabla J(\boldsymbol{w}_{t+1})^\top \boldsymbol{p}, \tag{2.11}$$

where the new estimate $\boldsymbol{B}_{t+1}$ of the inverse Hessian is computed in such a way that the gradient of $Q_{t+1}$ matches the gradient of $J$ at $\boldsymbol{w}_{t+1}$ and $\boldsymbol{w}_t$ (Figure 2.2), *i.e.*,

$$\nabla Q_{t+1}(\mathbf{0}) = \nabla J(\boldsymbol{w}_{t+1}) \;\text{ and }\; \nabla Q_{t+1}(-\eta_t \boldsymbol{p}_t) = \nabla J(\boldsymbol{w}_t). \tag{2.12}$$

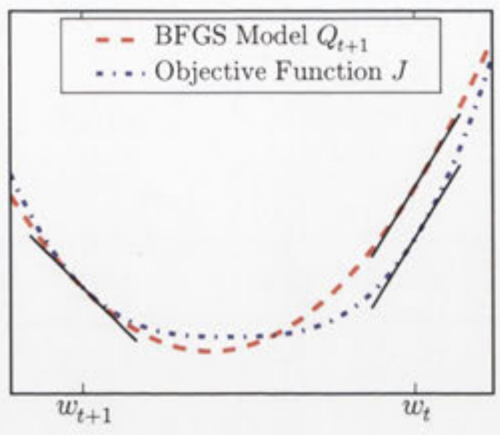


**Figure 2.2:** The gradient of the BFGS quadratic model $Q_{t+1}$ constructed at the new iterate $\boldsymbol{w}_{t+1}$ matches the gradient of the objective function $J$ at $\boldsymbol{w}_t$ and $\boldsymbol{w}_{t+1}$: tangents of $Q_{t+1}$ and $J$ (solid lines) are parallel at $\boldsymbol{w}_t$, and coincide at $\boldsymbol{w}_{t+1}$.

It is easy to check that $\nabla Q_{t+1}(\mathbf{0}) = \nabla J(\boldsymbol{w}_{t+1})$ holds independent of the exact form of $\boldsymbol{B}_{t+1}$. Using the derivative of (2.11) to expand the second equality in (2.12) gives

$$\nabla J(\boldsymbol{w}_{t+1}) - \eta_t \boldsymbol{B}_{t+1}^{-1} \boldsymbol{p}_t \; = \nabla J(\boldsymbol{w}_t). \tag{2.13}$$

Rearranging terms in (2.13) gives rise to the so-called *secant equation*:

$$\boldsymbol{B}_{t+1}\boldsymbol{y}_t = \boldsymbol{s}_t, \quad \text{where} \tag{2.14}$$

$$\boldsymbol{s}_t := \eta_t \boldsymbol{p}_t = \boldsymbol{w}_{t+1} - \boldsymbol{w}_t \quad \text{and} \quad \boldsymbol{y}_t := \nabla J(\boldsymbol{w}_{t+1}) - \nabla J(\boldsymbol{w}_t) \tag{2.15}$$

are the most recent steps along the optimization trajectory in parameter and gradient space, respectively. A matrix $\boldsymbol{B}_{t+1}$ that satisfies (2.14) is a good approximation to the inverse Hessian of the objective function. To see this, we use Taylor's theorem (Theorem 2.1.1 below) for the gradient of a continuously twice-differentiable function:

**Theorem 2.1.1** (Nocedal and Wright, 1999, Theorem 2.1) *Let $J : \mathbb{R}^d \to \mathbb{R}$ be continuously twice-differentiable and $\boldsymbol{p} \in \mathbb{R}^d$. Then we have*

$$\nabla J(\boldsymbol{w} + \boldsymbol{p}) = \nabla J(\boldsymbol{w}) + \int_0^1 \nabla^2 J(\boldsymbol{w} + t\boldsymbol{p})\, \boldsymbol{p}\, dt. \tag{2.16}$$

Simple manipulation of (2.16) gives

$$\nabla J(\boldsymbol{w} + \boldsymbol{p}) - \nabla J(\boldsymbol{w}) = \nabla^2 J(\boldsymbol{w})\, \boldsymbol{p} + \int_0^1 \left[\nabla^2 J(\boldsymbol{w} + t\boldsymbol{p}) - \nabla^2 J(\boldsymbol{w})\right] \boldsymbol{p}\, dt. \tag{2.17}$$

Further assume that the Hessian of $J$ is Lipschitz continuous,[1] then the size of the integral in (2.17) is $O(\|\boldsymbol{p}\|^2)$ (Dennis and Schnabel, 1996, Lemma 4.1.12). Omitting the higher order term, we can write

$$\nabla J(\boldsymbol{w}+\boldsymbol{p}) - \nabla J(\boldsymbol{w}) \approx \nabla^2 J(\boldsymbol{w})\,\boldsymbol{p}, \tag{2.18}$$

or equivalently

$$[\nabla^2 J(\boldsymbol{w})]^{-1}\,[\nabla J(\boldsymbol{w}+\boldsymbol{p}) - \nabla J(\boldsymbol{w})] \approx \boldsymbol{p}. \tag{2.19}$$

Comparing (2.19) with (2.14), we can see that $\boldsymbol{B}_{t+1}$ assumes the role of $[\nabla^2 J(\boldsymbol{w}_t)]^{-1}$.

Another implication of the secant equation (2.14) is that we must have

$$\boldsymbol{y}_t^\top \boldsymbol{B}_{t+1}\boldsymbol{y}_t = \boldsymbol{s}_t^\top \boldsymbol{y}_t > 0 \tag{2.20}$$

for any $\boldsymbol{s}_t, \boldsymbol{y}_t \neq \mathbf{0}$ since $\boldsymbol{B}_{t+1}$ must be positive definite. For strongly convex functions, Theorem 2.1.2 (below) shows that (2.20) is always true, provided that the norm of $\boldsymbol{s}_t$ is nonzero.

**Theorem 2.1.2** (Hiriart-Urruty and Lemaréchal, 1993, Theorem VI.6.1.2)
*A necessary and sufficient condition for a convex (and possibly nonsmooth) function $J : \mathbb{R}^d \to \mathbb{R}$ to be strongly convex (with modulus $c > 0$) on a convex set $\mathcal{C}$ is that the following inequality holds for all $\boldsymbol{w}_1, \boldsymbol{w}_2 \in \mathcal{C}$:*

$$(\boldsymbol{g}_2 - \boldsymbol{g}_1)^\top(\boldsymbol{w}_2 - \boldsymbol{w}_1) \geq c\,\|\boldsymbol{w}_2 - \boldsymbol{w}_1\|^2, \quad \textit{with} \;\; \boldsymbol{g}_i \in \partial J(\boldsymbol{w}_i),\; i = 1, 2. \tag{2.21}$$

On a general smooth convex function we can achieve this by using a line search that obeys the curvature condition (2.8). To see this, we rearrange (2.8) to write

$$[\nabla J(\boldsymbol{w}_{t+1}) - J(\boldsymbol{w}_t)]^\top \boldsymbol{p}_t \;\geq\; (c_2 - 1)\nabla J(\boldsymbol{w}_t)^\top \boldsymbol{p}_t \;>\; 0 \;\text{ with }\; c_2 \in (0,1), \tag{2.22}$$

where the last inequality holds because $\boldsymbol{p}_t$ satisfies the descent condition (2.4).

The secant equation (2.14) itself is not enough to determine $\boldsymbol{B}_{t+1}$ uniquely. Therefore, BFGS additionally requires $\boldsymbol{B}_{t+1}$ to be as close as possible to its previous iterate $\boldsymbol{B}_t$, in the sense that it minimizes a weighted Frobenius norm of the two. This results in the following constrained optimization problem in $\boldsymbol{B}_{t+1}$:

$$\begin{aligned} &\text{minimize}\,\|\boldsymbol{B}_{t+1} - \boldsymbol{B}_t\|_{\boldsymbol{W}} \\ &\text{s.t.}\;\; \boldsymbol{B}_{t+1}^\top = \boldsymbol{B}_{t+1},\;\; \boldsymbol{B}_{t+1}\boldsymbol{y}_t = \boldsymbol{s}_t,\;\; \text{and}\;\; \boldsymbol{W}\boldsymbol{s}_t = \boldsymbol{y}_t, \end{aligned} \tag{2.23}$$

[1]This means $\|\nabla^2 J(\boldsymbol{w}+\boldsymbol{p}) - \nabla^2 J(\boldsymbol{w})\| \leq c\|\boldsymbol{p}\|$ for some Lipschitz constant $c > 0$.

**Algorithm 2.2** Limited-Memory BFGS (LBFGS)

```
1: Initialize: $t := 0$ and $\boldsymbol{w}_0$
2: Set: convergence tolerance $\epsilon > 0$ and buffer size $m > 0$
3: while $\|\nabla J(\boldsymbol{w}_t)\| > \epsilon$ do
4:    Compute $\boldsymbol{p}_t$ via Algorithm 2.3
5:    Find $\eta_t$ that obeys (2.7) and (2.8)
6:    if $t > m$ then
7:       Discard vectors $\boldsymbol{s}_{t-m}$, $\boldsymbol{y}_{t-m}$
8:    end if
9:    Store $\boldsymbol{s}_t = \eta_t \boldsymbol{p}_t$
10:   $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \boldsymbol{s}_t$
11:   Store $\boldsymbol{y}_t = \nabla J(\boldsymbol{w}_{t+1}) - \nabla J(\boldsymbol{w}_t)$
12:   $t := t + 1$
13: end while
```

where the constraint on the weighting matrix $\boldsymbol{W}$ is justified by using an argument based on the invariant property of the weighted Frobenius norm (Fletcher, 1989, Thorem 3.3.2). The unique solution to (2.23) leads to a rank-two update[2] for $\boldsymbol{B}_{t+1}$:

$$\boldsymbol{B}_{t+1} = (\boldsymbol{I} - \rho_t \boldsymbol{s}_t \boldsymbol{y}_t^\top)\boldsymbol{B}_t(\boldsymbol{I} - \rho_t \boldsymbol{y}_t \boldsymbol{s}_t^\top) + \rho_t \boldsymbol{s}_t \boldsymbol{s}_t^\top, \quad \text{where} \quad \rho_t := (\boldsymbol{s}_t^\top \boldsymbol{y}_t)^{-1}. \tag{2.24}$$

The update (2.24) implicitly enforces the positivity of $\boldsymbol{B}_{t+1}$, provided that $\boldsymbol{B}_t$ is positive definite (Dennis and Schnabel, 1996, Thorem 7.8). Substantial experimental evidence has suggested that (2.24) yields the best inverse Hessian approximation, compared to other options such as DFP and SR1 approximations (Fletcher, 1989; Nocedal and Wright, 1999). In terms of practical implementation, the initial approximation $\boldsymbol{B}_0$ is usually set to the identity matrix, but subsequently scaled by an estimate of the largest eigenvalue of the inverse Hessian (Line 10 of Algorithm 2.1).

## 2.2 The Limited-Memory BFGS Method

Limited-memory BFGS (LBFGS, Algorithm 2.2) is a variant of BFGS designed for solving high-dimensional optimization problems where the $O(d^2)$ cost of storing and updating $\boldsymbol{B}_t$ would be prohibitive (Liu and Nocedal, 1989).

In LBFGS the estimation of the inverse Hessian is based on only the last $m$ steps in parameter and gradient space. Unrolling the recursive rank-two update (2.24), we

[2] The update (2.24) can be written as $\boldsymbol{B}_{t+1} = \boldsymbol{B}_t + \boldsymbol{C}$, where $\boldsymbol{C}$ is a rank-two correction matrix in the form $\boldsymbol{C} := \boldsymbol{a}\boldsymbol{b}^\top + \boldsymbol{b}\boldsymbol{a}^\top$; Fletcher (1989, Theorem 3.3.2) provides the exact forms of $\boldsymbol{a}$ and $\boldsymbol{b}$.

**Algorithm 2.3** LBFGS Direction Update

```
1: input buffer size $m > 0$, current iterate index $t \geq 0$, current gradient $\nabla J(\boldsymbol{w}_t)$,
       and $\forall\ i = 1, 2, \ldots, \min(t, m)$ : vectors $\boldsymbol{s}_{t-i}$ and $\boldsymbol{y}_{t-i}$ from Algorithm 2.2
2: output quasi-Newton direction $\boldsymbol{p}_t$
3: $\boldsymbol{p}_t := -\nabla J(\boldsymbol{w}_t)$
4: for $i := 1, 2, \ldots, \min(t, m)$ : do
5:     $\alpha_i = \frac{\boldsymbol{s}_{t-i}^\top \boldsymbol{p}_t}{\boldsymbol{s}_{t-i}^\top \boldsymbol{y}_{t-i}}$
6:     $\boldsymbol{p}_t := \boldsymbol{p}_t - \alpha_i \boldsymbol{y}_{t-i}$
7: end for
8: if $t > 0$ then
9:     $\boldsymbol{p}_t := \frac{\boldsymbol{s}_{t-1}^\top \boldsymbol{y}_{t-1}}{\boldsymbol{y}_{t-1}^\top \boldsymbol{y}_{t-1}} \boldsymbol{p}_t$
10: end if
11: for $i := \min(t, m), \ldots, 2, 1$ : do
12:     $\beta = \frac{\boldsymbol{y}_{t-i}^\top \boldsymbol{p}_t}{\boldsymbol{y}_{t-i}^\top \boldsymbol{s}_{t-i}}$
13:     $\boldsymbol{p}_t := \boldsymbol{p}_t + (\alpha_i - \beta) \boldsymbol{s}_{t-i}$
14: end for
15: return $\boldsymbol{p}_t$.
```

obtain the LBFGS inverse Hessian approximation:

$$\begin{aligned}
\boldsymbol{B}_{t+1} = {} & [\boldsymbol{A}_t^\top \cdots \boldsymbol{A}_{t-m+1}^\top]\ \boldsymbol{B}_0\ [\boldsymbol{A}_{t-m+1} \cdots \boldsymbol{A}_t] \\
& + \rho_{t-m+1}\ [\boldsymbol{A}_t^\top \cdots \boldsymbol{A}_{t-m+2}^\top]\ \boldsymbol{s}_{t-m+1} \boldsymbol{s}_{t-m+1}^\top\ [\boldsymbol{A}_{t-m+2} \cdots \boldsymbol{A}_t] \\
& + \rho_{t-m+2}\ [\boldsymbol{A}_t^\top \cdots \boldsymbol{A}_{t-m+3}^\top]\ \boldsymbol{s}_{t-m+2} \boldsymbol{s}_{t-m+2}^\top\ [\boldsymbol{A}_{t-m+3} \cdots \boldsymbol{A}_t] \\
& + \cdots \\
& + \rho_t \boldsymbol{s}_t^\top \boldsymbol{s}_t,
\end{aligned} \tag{2.25}$$

where the auxiliary matrix $\boldsymbol{A}_t$ is defined as $\boldsymbol{A}_t := (\boldsymbol{I} - \rho_t \boldsymbol{y}_t \boldsymbol{s}_t^\top)$. It is customary to set the initial approximation $\boldsymbol{B}_0$ to a scaled identity matrix, but unlike in BFGS where the scaling factor is fixed, here it can vary from iteration to iteration to reflect the latest estimate of the largest eigenvalue of the inverse Hessian. Nocedal (1980) shows that the product of a matrix of the form (2.25) with a vector can be efficiently computed via a recursive procedure. Algorithm 2.3 implements this procedure to obtain the quasi-Newton direction $-\boldsymbol{B}_t \nabla J(\boldsymbol{w}_t)$. Note that $\boldsymbol{B}_t$ is not explicitly used by Algorithm 2.3. A standard implementation of LBFGS (Algorithm 2.2) thus omits Lines 4 and 9–13 from Algorithm 2.1, maintains a buffer of the last $m$ parameter and gradient displacement vectors, *i.e.*, $\forall\ i = 1, 2, \ldots, \min(t, m)$ : vectors $\boldsymbol{s}_{t-i}$ and $\boldsymbol{y}_{t-i}$ as in (2.15), and replaces Line 4 of Algorithm 2.1 with Algorithm 2.3. This reduces the cost from $O(d^2)$ to $O(md)$ space and time per iteration, with $m$ freely chosen (typically between 3 and 20).

## 2.3 Summary

We reviewed the standard BFGS quasi-Newton method and its limited-memory variant (LBFGS). These two quasi-Newton optimizers are widely considered as the workhorses of *smooth* nonlinear optimization due to their superior practical performance. However, their application to nonsmooth optimization has been problematic because their key components critically depend on differentiability of the objective function. In the next chapter we relax this dependence so as to generalize this framework from smooth to nonsmooth nonlinear optimization. In Chapter 5 we then extend (L)BFGS to the stochastic setting where optimization is based on approximate function (*resp.* gradient) measurements obtained from small subsamples of the training data.

Chapter 3

# A Quasi-Newton Approach to Nonsmooth Convex Optimization

In this chapter we extend the classical BFGS method to nonsmooth convex optimization. This is done in a rigorous fashion by generalizing three components of BFGS to subdifferentials: the local quadratic model, the identification of a descent direction, and the Wolfe line search conditions. We prove that under some technical conditions, the resulting subBFGS algorithm is globally convergent in objective function value. We demonstrate the use of our algorithms for $L_2$-regularized risk minimization with the hinge loss. To extend them to the multiclass and multilabel settings, we also develop a new, efficient, exact line search algorithm. Throughout this chapter we assume that the objective function $J : \mathbb{R}^d \rightarrow \mathbb{R}$ is convex.

We first motivate our work by illustrating the difficulties of (L)BFGS on nonsmooth functions, and the advantage of incorporating BFGS' curvature estimate into the parameter update. In Section 3.2 we develop our optimization algorithms generically, before discussing their application to $L_2$-regularized risk minimization with the hinge loss in Section 3.3. We describe a new efficient algorithm to identify the nonsmooth points of a one-dimensional pointwise maximum of linear functions in Section 3.4, then use it to develop an exact line search that extends our optimization algorithms to the multiclass and multilabel settings (Section 3.5). We compare and contrast our work with other recent efforts in this area in Section 3.6, before concluding this chapter with a discussion (Section 3.7). Our experimental results on a number of public machine learning datasets are presented in Chapter 4.

## 3.1 Motivation

BFGS (*resp.* LBFGS) works surprisingly well on some nonsmooth problems but is not guaranteed to converge (Haarala, 2004; Lewis and Overton, 2008a,b; Lukšan and Vlček, 1999). Various fixes can be used to avoid this problem, but only in an ad-hoc manner. Therefore, subgradient-based approaches such as subgradient descent (Nedić and

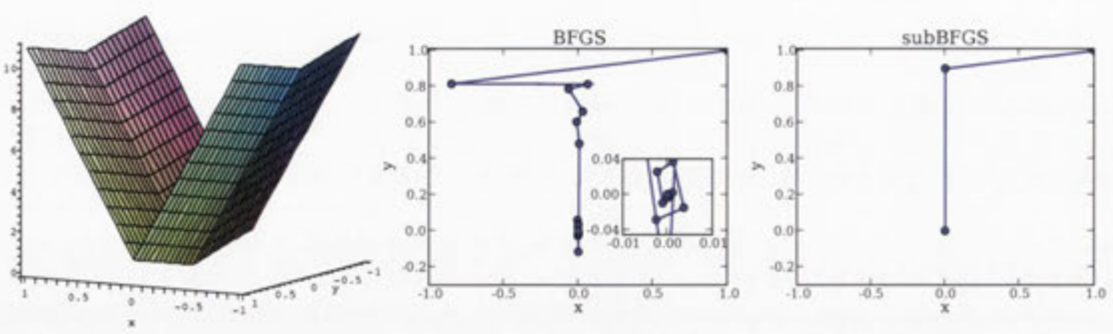


**Figure 3.1:** Left: the nonsmooth convex function (3.1); optimization trajectory of BFGS with inexact line search (center) and subBFGS (right) on this function.

Bertsekas, 2000) or bundle methods (Franc and Sonnenburg, 2008; Joachims, 2006; Teo et al., 2010) have gained considerable attention for minimizing nonsmooth objectives. Our aim is to develop principled and robust quasi-Newton methods that are suitable for solving nonsmooth convex optimization problems in machine learning.

The application of standard (L)BFGS to nonsmooth optimization has been problematic since the quasi-Newton direction generated at a nonsmooth point is not necessarily a descent direction. Nevertheless, BFGS' inverse Hessian estimate can still be used to effectively model the shape of a nonsmooth objective; incorporating it into the parameter update can therefore be beneficial. We discuss these two aspects of (L)BFGS to motivate our work on developing new quasi-Newton methods that are amenable to subgradients while preserving the fast convergence properties of standard (L)BFGS.

### 3.1.1 Problems of (L)BFGS on Nonsmooth Objectives

Smoothness of the objective function is essential for classical (L)BFGS because both the local quadratic model (2.1) and the Wolfe conditions (2.7, 2.8) require the existence of the gradient $\nabla J$ at every point. As pointed out by Hiriart-Urruty and Lemaréchal (1993, Remark VIII.2.1.3), even though nonsmooth convex functions are differentiable everywhere except on a set of Lebesgue measure zero, it is unwise to just use a smooth optimizer on a nonsmooth convex problem under the assumption that "it should work almost surely." Below we illustrate this on both a toy example and real-world machine learning problems.

#### 3.1.1.1 A Toy Example

The following simple example demonstrates the problems faced by BFGS when working with a nonsmooth objective function, and how our subgradient BFGS (subBFGS) method (to be introduced in Section 3.2) with exact line search overcomes these problems. Consider the task of minimizing

$$f(x, y) = 10\,|x| + |y| \tag{3.1}$$

with respect to $x$ and $y$. Clearly, $f(x, y)$ is convex but nonsmooth, with the minimum located at $(0, 0)$ (Figure 3.1, left). It is subdifferentiable whenever $x$ or $y$ is zero:

$$\partial_x f(0, \cdot) = [-10, 10] \text{ and } \partial_y f(\cdot, 0) = [-1, 1]. \tag{3.2}$$

We call such lines of subdifferentiability in parameter space *hinges*.

We can minimize (3.1) with the standard BFGS algorithm, employing a backtracking line search (Nocedal and Wright, 1999, Procedure 3.1) that starts with a step size that obeys the curvature condition (2.8), then exponentially decays it until both Wolfe conditions (2.7, 2.8) are satisfied.[1] The curvature condition forces BFGS to jump across at least one hinge, thus ensuring that the gradient displacement vector $\boldsymbol{y}_t$ in (2.24) is non-zero; this prevents BFGS from diverging. Moreover, with such an *inexact* line search BFGS will generally not step on any hinges directly, thus avoiding (in an ad-hoc manner) the problem of non-differentiability. Although this algorithm quickly decreases the objective from the starting point $(1, 1)$, it is then slowed down by heavy oscillations around the optimum (Figure 3.1, center), caused by the utter mismatch between BFGS' quadratic model and the actual function.

A generally sensible strategy is to use an exact line search that finds the optimum along a given descent direction (*cf.* Section 3.3.2.1). However, this line optimum will often lie on a hinge (as it does in our toy example), where the function is not differentiable. If an arbitrary subgradient is supplied instead, the BFGS update (2.24) can produce a search direction which is not a descent direction, causing the next line search to fail. In our toy example, standard BFGS with exact line search consistently fails after the first step, which takes it to the hinge at $x = 0$.

Unlike standard BFGS, our subBFGS method can handle hinges and thus reap the benefits of an exact line search. As Figure 3.1 (right) shows, once the first iteration of subBFGS lands it on the hinge at $x = 0$, its direction-finding routine (Algorithm 3.2) finds a descent direction for the next step. In fact, on this simple example Algorithm 3.2 yields a vector with zero $x$ component, which takes subBFGS straight to the optimum at the second step.[2]

#### 3.1.1.2 Typical Nonsmooth Optimization Problems in Machine Learning

The problems faced by smooth quasi-Newton methods on nonsmooth objectives are not only encountered in cleverly constructed toy examples, but also in real-world appli-

[1]We set $c_1 = 10^{-3}$ in (2.7) and $c_2 = 0.8$ in (2.8), and used a decay factor of 0.9.

[2]This is achieved for any choice of initial subgradient $\boldsymbol{g}^{(1)}$ (Line 3 of Algorithm 3.2).

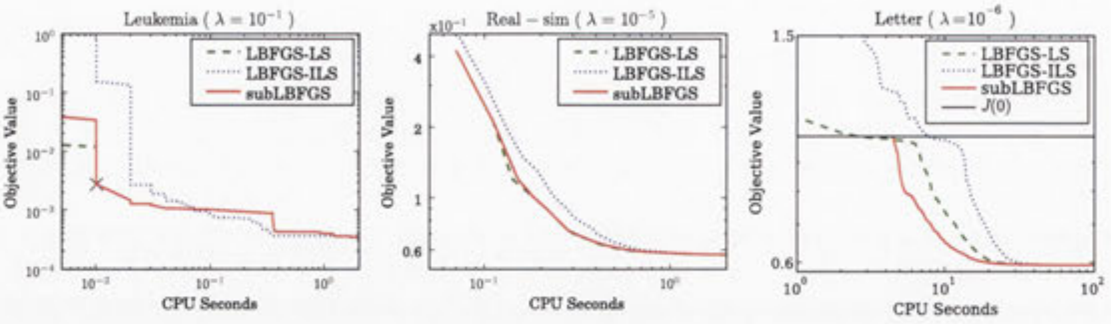


**Figure 3.2:** Performance of subLBFGS (solid) and standard LBFGS with exact (dashed) and inexact (dotted) line search methods on sample $L_2$-regularized risk minimization problems with the binary (left and center) and multiclass hinge losses (right). LBFGS with exact line search (dashed) fails after 3 iterations (marked as $\times$) on the Leukemia dataset (left).

cations. To show this, we apply LBFGS to $L_2$-regularized risk minimization problems (1.1) with binary hinge loss (1.2), a typical nonsmooth optimization problem encountered in machine learning. For this particular objective function, an exact line search is cheap and easy to compute (see Section 3.3.2.1 for details). Figure 3.2 (left & center) shows the behavior of LBFGS with this exact line search (LBFGS-LS) on two datasets, namely Leukemia and Real-sim.[3] It can be seen that LBFGS-LS converges on Real-sim but diverges on the Leukemia dataset. This is because using an exact line search on a nonsmooth objective function increases the chance of landing on nonsmooth points, a situation that standard BFGS (*resp.* LBFGS) is not designed to deal with. To prevent (L)BFGS' sudden breakdown, a scheme that actively avoids nonsmooth points must be used. One such possibility is to use an inexact line search that obeys the Wolfe conditions. Here we used an efficient inexact line search that uses a caching scheme specifically designed for $L_2$-regularized hinge loss (*cf.* end of Section 3.3.2). This implementation of LBFGS (LBFGS-ILS) converges on both datasets shown here but may fail on others. It is also slower, due to the inexactness of its line search.

For the multiclass hinge loss (3.40) we encounter another problem: if we follow the usual practice of initializing $\boldsymbol{w} = \mathbf{0}$, which happens to be a non-differentiable point, then LBFGS stalls. One way to get around this is to force LBFGS to take a unit step along its search direction to escape this nonsmooth point. However, as can be seen on the Letter dataset[3] in Figure 3.2 (right), such an ad-hoc fix increases the value of the objective above $J(\mathbf{0})$ (solid horizontal line), and it takes several CPU seconds for the optimizers to recover from this. In all cases shown in Figure 3.2, our subgradient LBFGS (subLBFGS) method (as will be introduced later) performs comparable to or better than the best implementation of LBFGS.

[3]Descriptions of these datasets can be found in Section 4.1.

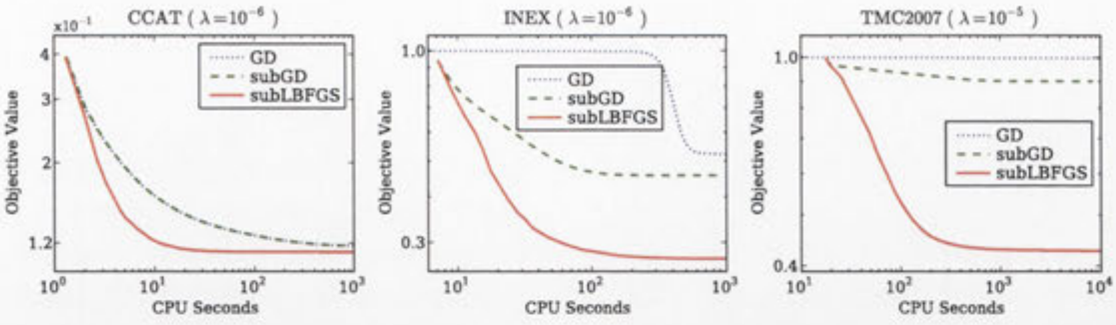


**Figure 3.3:** Performance of subLBFGS, GD, and subGD on sample $L_2$-regularized risk minimization problems with the binary (left), multiclass (center), and multilabel (right) hinge losses.

### 3.1.2 Advantage of Incorporating BFGS' Curvature Estimate

In machine learning one often encounters $L_2$-regularized risk minimization problems (1.1) with various hinge losses (1.2, 3.40, 3.55). Since the Hessian of those objective functions at differentiable points equals $\lambda \boldsymbol{I}$ (where $\lambda$ is the regularization constant), one might be tempted to argue that for such problems, BFGS' approximation $\boldsymbol{B}_t$ to the inverse Hessian should be simply set to $\lambda^{-1}\boldsymbol{I}$. This would reduce the quasi-Newton direction $\boldsymbol{p}_t = -\boldsymbol{B}_t\boldsymbol{g}_t,\ \boldsymbol{g}_t \in \partial J(\boldsymbol{w}_t)$ to simply a scaled subgradient direction.

To check if doing so is beneficial, we compared the performance of our subLBFGS method with two implementations of subgradient descent: a vanilla gradient descent method (denoted GD) that uses a random subgradient for its parameter update, and an improved subgradient descent method (denoted subGD) whose parameter is updated in the direction produced by our direction-finding routine (Algorithm 3.2) with $\boldsymbol{B}_t = \boldsymbol{I}$. All algorithms used exact line search, except that GD took a unit step for the first update in order to avoid the nonsmooth point $\boldsymbol{w}_0 = \boldsymbol{0}$ (*cf.* the discussion in Section 3.1.1.2). As can be seen in Figure 3.3, on all sample $L_2$-regularized hinge loss minimization problems, subLBFGS (solid) converges significantly faster than GD (dotted) and subGD (dashed). This indicates that BFGS' $\boldsymbol{B}_t$ matrix is able to model the objective function, including its hinges, better than simply setting $\boldsymbol{B}_t$ to a scaled identity matrix.

We believe that BFGS' curvature update (2.24) plays an important role in the performance of subLBFGS seen in Figure 3.3. Recall that (2.24) satisfies the secant condition $\boldsymbol{B}_{t+1}\boldsymbol{y}_t = \boldsymbol{s}_t$, where $\boldsymbol{s}_t$ and $\boldsymbol{y}_t$ are displacement vectors in parameter and gradient space, respectively. The secant condition in fact implements a *finite differencing*

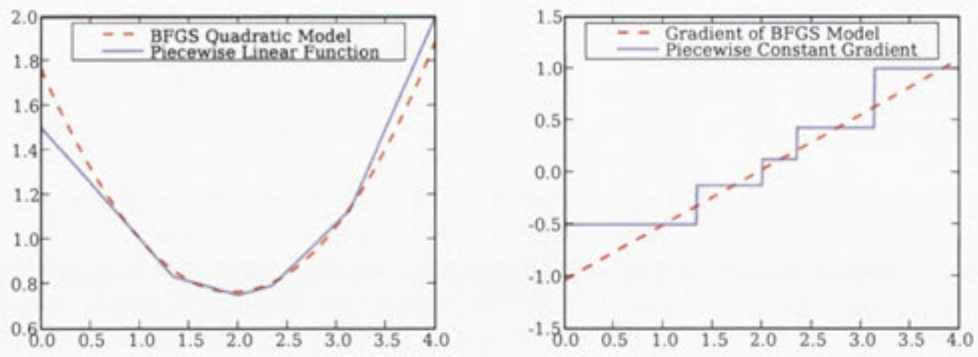


**Figure 3.4:** BFGS' quadratic approximation to a piecewise linear function (left), and its estimate of the gradient of this function (right).

scheme: for a one-dimensional objective function $J : \mathbb{R} \to \mathbb{R}$, we have

$$\boldsymbol{B}_{t+1} = \frac{(w+p) - w}{\nabla J(w+p) - \nabla J(w)}. \tag{3.3}$$

Although the original motivation behind the secant condition was to approximate the inverse Hessian, the finite differencing scheme (3.3) allows BFGS to model the global curvature (*i.e.*, overall shape) of the objective function from first-order information. For instance, Figure 3.4 (left) shows that the BFGS quadratic model[4] (2.1) fits a piecewise linear function quite well despite the fact that the actual Hessian in this case is zero almost everywhere, and infinite (in the limit) at nonsmooth points. Figure 3.4 (right) reveals that BFGS captures the global trend of the gradient rather than its infinitesimal variation, that is, the Hessian. This is beneficial for nonsmooth problems, where Hessian does not fully represent the overall curvature of the objective function.

## 3.2 Subgradient BFGS Method

We modify the standard BFGS algorithm to derive our new algorithm (subBFGS, Algorithm 3.1) for nonsmooth convex optimization. Our modifications can be grouped into three areas, which we elaborate on in turn: generalizing the local quadratic model, finding a descent direction, and finding a step size that obeys a subgradient reformulation of the Wolfe conditions. We then show that our algorithm's estimate of the inverse Hessian has a bounded spectrum, which allows us to prove its convergence.

[4]For ease of exposition, the model was constructed at a differentiable point.

**Algorithm 3.1** Subgradient BFGS (subBFGS)

```
1: Initialize: $t := 0, \boldsymbol{w}_0 = \boldsymbol{0}, \boldsymbol{B}_0 = \boldsymbol{I}$
2: Set: direction-finding tolerance $\epsilon \geq 0$, iteration limit $k_{\max} > 0$,
        lower bound $h > 0$ on $\frac{\boldsymbol{s}_t^\top \boldsymbol{y}_t}{\boldsymbol{y}_t^\top \boldsymbol{y}_t}$ (cf. discussion in Section 3.2.4)
3: Compute subgradient $\boldsymbol{g}_0 \in \partial J(\boldsymbol{w}_0)$
4: while not converged do
5:    $\boldsymbol{p}_t = \texttt{descentDirection}(\boldsymbol{g}_t, \epsilon, k_{\max})$          (Algorithm 3.2)
6:    if $\boldsymbol{p}_t$ = failure then
7:       Return $\boldsymbol{w}_t$
8:    end if
9:    Find $\eta_t$ that obeys (3.17) and (3.18)          (e.g., Algorithm 3.3 or 3.5)
10:   $\boldsymbol{s}_t = \eta_t \boldsymbol{p}_t$
11:   $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \boldsymbol{s}_t$
12:   Choose subgradient $\boldsymbol{g}_{t+1} \in \partial J(\boldsymbol{w}_{t+1}) : \boldsymbol{s}_t^\top(\boldsymbol{g}_{t+1} - \boldsymbol{g}_t) > 0$
13:   $\boldsymbol{y}_t := \boldsymbol{g}_{t+1} - \boldsymbol{g}_t$
14:   $\boldsymbol{s}_t := \boldsymbol{s}_t + \max\left(0,\ h - \frac{\boldsymbol{s}_t^\top \boldsymbol{y}_t}{\boldsymbol{y}_t^\top \boldsymbol{y}_t}\right) \boldsymbol{y}_t$          (ensure $\frac{\boldsymbol{s}_t^\top \boldsymbol{y}_t}{\boldsymbol{y}_t^\top \boldsymbol{y}_t} \geq h$)
15:   Update $\boldsymbol{B}_{t+1}$ via (2.24)
16:   $t := t + 1$
17: end while
```

### 3.2.1 Generalizing the Local Quadratic Model

Recall that BFGS assumes that the objective function $J$ is differentiable everywhere so that at the current iterate $\boldsymbol{w}_t$ it can construct a local quadratic model (2.1) of $J(\boldsymbol{w}_t)$. For a nonsmooth objective function, such a model becomes ambiguous at non-differentiable points (Figure 3.5, left). To resolve the ambiguity, we could simply replace the gradient $\nabla J(\boldsymbol{w}_t)$ in (2.1) with an arbitrary subgradient $\boldsymbol{g}_t \in \partial J(\boldsymbol{w}_t)$. However, as will be discussed later, the resulting quasi-Newton direction $\boldsymbol{p}_t := -\boldsymbol{B}_t \boldsymbol{g}_t$ is not necessarily a descent direction. To address this fundamental modeling problem, we first generalize the local quadratic model (2.1) as follows:

$$\begin{aligned} Q_t(\boldsymbol{p}) &:= J(\boldsymbol{w}_t) + M_t(\boldsymbol{p}), \quad \text{where} \\ M_t(\boldsymbol{p}) &:= \tfrac{1}{2}\boldsymbol{p}^\top \boldsymbol{B}_t^{-1} \boldsymbol{p} + \sup_{\boldsymbol{g} \in \partial J(\boldsymbol{w}_t)} \boldsymbol{g}^\top \boldsymbol{p}. \end{aligned} \tag{3.4}$$

Note that where $J$ is differentiable, (3.4) reduces to the familiar BFGS quadratic model (2.1). At non-differentiable points, however, the model is no longer quadratic, as the supremum may be attained at different elements of $\partial J(\boldsymbol{w}_t)$ for different directions $\boldsymbol{p}$. Instead it can be viewed as the tightest pseudo-quadratic fit to $J$ at $\boldsymbol{w}_t$ (Figure 3.5, right).

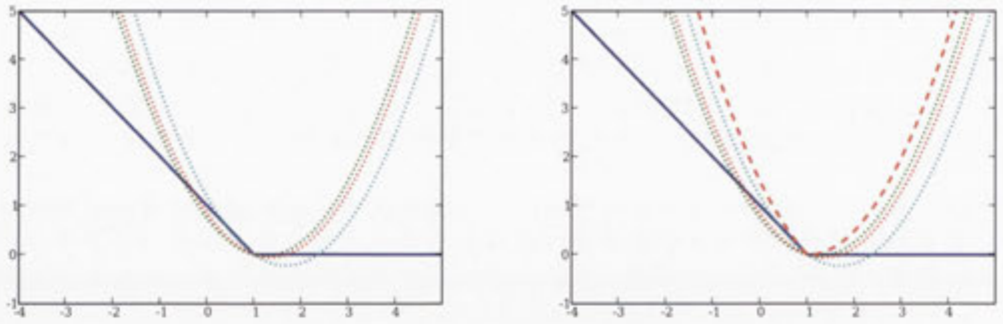


**Figure 3.5:** Left: selecting arbitrary subgradients yields many possible quadratic models (dotted lines) for the objective (solid blue line) at a subdifferentiable point. The models were built by keeping $\boldsymbol{B}_t$ fixed, but selecting random subgradients. Right: the tightest pseudo-quadratic fit (3.4) (bold red dashes); note that it is not a quadratic.

Having constructed the model (3.4), we can minimize $Q_t(\boldsymbol{p})$, or equivalently $M_t(\boldsymbol{p})$:

$$\min_{\boldsymbol{p}\in\mathbb{R}^d} \left( \tfrac{1}{2}\boldsymbol{p}^\top \boldsymbol{B}_t^{-1}\boldsymbol{p} + \sup_{\boldsymbol{g}\in\partial J(\boldsymbol{w}_t)} \boldsymbol{g}^\top \boldsymbol{p} \right) \tag{3.5}$$

to obtain a search direction. We now show that solving (3.5) is closely related to the problem of finding a *normalized steepest descent* direction. A normalized steepest descent direction is defined as the solution to the following problem (Hiriart-Urruty and Lemaréchal, 1993, Chapter VIII):

$$\min_{\boldsymbol{p}\in\mathbb{R}^d} \; J'(\boldsymbol{w}_t,\, \boldsymbol{p}) \;\text{ s.t. }\; \|\boldsymbol{p}\| \le 1, \tag{3.6}$$

where

$$J'(\boldsymbol{w}_t,\, \boldsymbol{p}) := \lim_{\eta\downarrow 0} \frac{J(\boldsymbol{w}_t + \eta\boldsymbol{p}) - J(\boldsymbol{w}_t)}{\eta}$$

is the directional derivative of $J$ at $\boldsymbol{w}_t$ in direction $\boldsymbol{p}$, and $\|\cdot\|$ is a norm defined on $\mathbb{R}^d$. In other words, the normalized steepest descent direction is the direction of bounded norm along which the maximum rate of decrease in the objective function value is achieved. Using the property: $J'(\boldsymbol{w}_t,\, \boldsymbol{p}) = \sup_{\boldsymbol{g}\in\partial J(\boldsymbol{w}_t)} \boldsymbol{g}^\top \boldsymbol{p}$ (Bertsekas, 1999, Proposition B.24.b), we can rewrite (3.6) as:

$$\min_{\boldsymbol{p}\in\mathbb{R}^d} \; \sup_{\boldsymbol{g}\in\partial J(\boldsymbol{w}_t)} \boldsymbol{g}^\top \boldsymbol{p} \;\text{ s.t. }\; \|\boldsymbol{p}\| \le 1. \tag{3.7}$$

If the matrix $\boldsymbol{B}_t \succ 0$ as in (3.5) is used to define the norm $\|\cdot\|$ as

$$\|\boldsymbol{p}\|^2 := \boldsymbol{p}^\top \boldsymbol{B}_t^{-1}\boldsymbol{p}, \tag{3.8}$$

then the solution to (3.7) points to the same direction as that obtained by minimizing our pseudo-quadratic model (3.5). To see this, we write the Lagrangian of the constrained minimization problem (3.7):

$$\begin{aligned} L(\boldsymbol{p}, \alpha) &:= \alpha\, \boldsymbol{p}^\top \boldsymbol{B}_t^{-1} \boldsymbol{p} \,-\alpha \,+ \sup_{\boldsymbol{g}\in\partial J(\boldsymbol{w}_t)} \boldsymbol{g}^\top \boldsymbol{p} \\ &= \tfrac{1}{2}\boldsymbol{p}^\top (2\alpha\, \boldsymbol{B}_t^{-1})\boldsymbol{p} \,-\alpha \,+ \sup_{\boldsymbol{g}\in\partial J(\boldsymbol{w}_t)} \boldsymbol{g}^\top \boldsymbol{p}, \end{aligned} \tag{3.9}$$

where $\alpha \geq 0$ is a Lagrangian multiplier. It is easy to see from (3.9) that minimizing the Lagrangian function $L$ with respect to $\boldsymbol{p}$ is equivalent to solving (3.5) with $\boldsymbol{B}_t^{-1}$ scaled by a scalar $2\alpha$, implying that the steepest descent direction obtained by solving (3.7) with the weighted norm (3.8) only differs in length from the search direction obtained by solving (3.5). Therefore, our search direction is essentially an unnomalized steepest descent direction with respect to the weighted norm (3.8).

Ideally, we would like to solve (3.5) to obtain the best search direction. This is generally intractable due to the presence a supremum over the entire subdifferential set $\partial J(\boldsymbol{w}_t)$. In many machine learning problems, however, $\partial J(\boldsymbol{w}_t)$ has some special structure that simplifies the calculation of that supremum. In particular, the subdifferential of all the problems considered in this chapter is a convex and compact polyhedron characterised as the convex hull of its extreme points. This dramatically reduces the cost of calculating $\sup_{\boldsymbol{g}\in\partial J(\boldsymbol{w}_t)} \boldsymbol{g}^\top \boldsymbol{p}$ since the supremum can only be attained at an extreme point of the polyhedral set $\partial J(\boldsymbol{w}_t)$ (Bertsekas, 1999, Proposition B.21c). In what follows, we develop an iterative procedure that is guaranteed to find a quasi-Newton descent direction, assuming an oracle that supplies $\arg\sup_{\boldsymbol{g}\in\partial J(\boldsymbol{w}_t)} \boldsymbol{g}^\top \boldsymbol{p}$ for a given direction $\boldsymbol{p} \in \mathbb{R}^d$. Efficient oracles for this purpose can be derived for many machine learning settings; we provides such oracles for $L_2$-regularized risk minimization with the binary hinge loss (Section 3.3.1), multiclass and multilabel hinge losses (Section 3.5), and $L_1$-regularized logistic loss (Section 4.1.4).

### 3.2.2 Finding a Descent Direction

A direction $\boldsymbol{p}_t$ is a descent direction if and only if $\boldsymbol{g}^\top \boldsymbol{p}_t < 0 \;\; \forall \boldsymbol{g} \in \partial J(\boldsymbol{w}_t)$ (Hiriart-Urruty and Lemaréchal, 1993, Theorem VIII.1.1.2), or equivalently

$$\sup_{\boldsymbol{g}\in\partial J(\boldsymbol{w}_t)} \boldsymbol{g}^\top \boldsymbol{p}_t \;<\; 0. \tag{3.10}$$

For a smooth convex function, the quasi-Newton direction (2.3) is always a descent direction because

$$\nabla J(\boldsymbol{w}_t)^\top \boldsymbol{p}_t \;=\; -\nabla J(\boldsymbol{w}_t)^\top \boldsymbol{B}_t \nabla J(\boldsymbol{w}_t) \;<\; 0$$

**Algorithm 3.2** $\boldsymbol{p}_t = \texttt{descentDirection}(\boldsymbol{g}^{(1)}, \epsilon, k_{\max})$

```
1: input (sub)gradient $g^{(1)} \in \partial J(w_t)$, tolerance $\epsilon \geq 0$, iteration limit $k_{\max} > 0$,
       and an oracle to calculate $\arg\sup_{g \in \partial J(w)} g^\top p$ for any given $w$ and $p$
2: output descent direction $p_t$
3: Initialize: $i = 1,\ \bar{g}^{(1)} = g^{(1)},\ p^{(1)} = -B_t g^{(1)}$
4: $g^{(2)} = \arg\sup_{g \in \partial J(w_t)} g^\top p^{(1)}$
5: $\epsilon^{(1)} := p^{(1)\top} g^{(2)} - p^{(1)\top} \bar{g}^{(1)}$
6: while $(g^{(i+1)\top} p^{(i)} > 0$ or $\epsilon^{(i)} > \epsilon)$ and $\epsilon^{(i)} > 0$ and $i < k_{\max}$ do
7:    $\mu^* := \min\left[1, \frac{(\bar{g}^{(i)} - g^{(i+1)})^\top B_t \bar{g}^{(i)}}{(\bar{g}^{(i)} - g^{(i+1)})^\top B_t (\bar{g}^{(i)} - g^{(i+1)})}\right]$;  cf. (A.43)
8:    $\bar{g}^{(i+1)} = (1 - \mu^*) \bar{g}^{(i)} + \mu^* g^{(i+1)}$
9:    $p^{(i+1)} = (1 - \mu^*) p^{(i)} - \mu^* B_t g^{(i+1)}$;  cf. (A.18)
10:   $g^{(i+2)} = \arg\sup_{g \in \partial J(w_t)} g^\top p^{(i+1)}$
11:   $\epsilon^{(i+1)} := \min_{j \leq (i+1)} \left[p^{(j)\top} g^{(j+1)} - \frac{1}{2}(p^{(j)\top} \bar{g}^{(j)} + p^{(i+1)\top} \bar{g}^{(i+1)})\right]$
12:   $i := i + 1$
13: end while
14: $p_t = \operatorname{argmin}_{j \leq i} M_t(p^{(j)})$
15: if $\sup_{g \in \partial J(w_t)} g^\top p_t \geq 0$ then
16:    return failure;
17: else
18:    return $p_t$.
19: end if
```

holds due to the positivity of $\boldsymbol{B}_t$.

For nonsmooth functions, however, the quasi-Newton direction $\boldsymbol{p}_t := -\boldsymbol{B}_t \boldsymbol{g}_t$ for a given $\boldsymbol{g}_t \in \partial J(\boldsymbol{w}_t)$ may not fulfill the descent condition (3.10), making it impossible to find a step size $\eta > 0$ that obeys the Wolfe conditions (2.7, 2.8), thus causing a failure of the line search. We now present an iterative approach to finding a quasi-Newton *descent* direction.

Our goal is to minimize the pseudo-quadratic model (3.4), or equivalently minimize $M_t(\boldsymbol{p})$. Inspired by bundle methods (Teo et al., 2010), we achieve this by minimizing convex lower bounds of $M_t(\boldsymbol{p})$ that are designed to progressively approach $M_t(\boldsymbol{p})$ over iterations. At iteration $i$ we build the following convex lower bound on $M_t(\boldsymbol{p})$:

$$M_t^{(i)}(\boldsymbol{p}) \;:=\; \tfrac{1}{2}\boldsymbol{p}^\top \boldsymbol{B}_t^{-1} \boldsymbol{p} + \sup_{j \leq i} \boldsymbol{g}^{(j)\top} \boldsymbol{p}, \tag{3.11}$$

where $i, j \in \mathbb{N}$ and $\boldsymbol{g}^{(j)} \in \partial J(\boldsymbol{w}_t)\ \forall j \leq i$. Given a $\boldsymbol{p}^{(i)} \in \mathbb{R}^d$ the lower bound (3.11) is successively tightened by computing

$$\boldsymbol{g}^{(i+1)} := \operatorname*{arg\,sup}_{\boldsymbol{g} \in \partial J(\boldsymbol{w}_t)} \boldsymbol{g}^\top \boldsymbol{p}^{(i)}, \tag{3.12}$$

such that $M_t^{(i)}(\boldsymbol{p}) \leq M_t^{(i+1)}(\boldsymbol{p}) \leq M_t(\boldsymbol{p})\ \forall \boldsymbol{p} \in \mathbb{R}^d$. Here we set $\boldsymbol{g}^{(1)} \in \partial J(\boldsymbol{w}_t)$ arbitrarily, and assume that (3.12) is provided by an oracle (*e.g.*, as described in Section 3.3.1). To solve $\min_{\boldsymbol{p}\in\mathbb{R}^d} M_t^{(i)}(\boldsymbol{p})$, we rewrite it as a constrained optimization problem:

$$\min_{\boldsymbol{p},\xi} \left(\tfrac{1}{2}\boldsymbol{p}^\top \boldsymbol{B}_t^{-1}\boldsymbol{p} + \xi\right) \ \text{ s.t. } \ \boldsymbol{g}^{(j)\top}\boldsymbol{p} \leq \xi \ \ \forall j \leq i. \tag{3.13}$$

This problem can be solved exactly via quadratic programming, but doing so may incur substantial computational expense. Instead we adopt an alternative approach (Algorithm 3.2) which does not solve (3.13) to optimality. The key idea is to write the proposed descent direction at iteration $i+1$ as a convex combination of $\boldsymbol{p}^{(i)}$ and $-\boldsymbol{B}_t\boldsymbol{g}^{(i+1)}$ (Line 9 of Algorithm 3.2); and as will be shown in Appendix A.2, the returned search direction takes the form

$$\boldsymbol{p}_t = -\boldsymbol{B}_t\bar{\boldsymbol{g}}_t, \tag{3.14}$$

where $\bar{\boldsymbol{g}}_t$ is a subgradient in $\partial J(\boldsymbol{w}_t)$ that allows $\boldsymbol{p}_t$ to satisfy the descent condition (3.10). The optimal convex combination coefficient $\mu^*$ can be computed exactly (Line 7 of Algorithm 3.2) using an argument based on maximizing the dual objective of $M_t(\boldsymbol{p})$; see Appendix A.1 for details.

The weak duality theorem (Hiriart-Urruty and Lemaréchal, 1993, Theorem XII.2.1.5) states that the optimal primal value is no less than any dual value, *i.e.*, if $D_t(\boldsymbol{\alpha})$ is the dual of $M_t(\boldsymbol{p})$, then $\min_{\boldsymbol{p}\in\mathbb{R}^d} M_t(\boldsymbol{p}) \geq D_t(\boldsymbol{\alpha})$ holds for all feasible dual solutions $\boldsymbol{\alpha}$. Therefore, by iteratively increasing the value of the dual objective we close the gap to optimality in the primal. Based on this argument, we use the following upper bound on the duality gap as our measure of progress:

$$\epsilon^{(i)} := \min_{j\leq i}\left[\boldsymbol{p}^{(j)\top}\boldsymbol{g}^{(j+1)} - \tfrac{1}{2}(\boldsymbol{p}^{(j)\top}\bar{\boldsymbol{g}}^{(j)} + \boldsymbol{p}^{(i)\top}\bar{\boldsymbol{g}}^{(i)})\right] \geq \min_{\boldsymbol{p}\in\mathbb{R}^d} M_t(\boldsymbol{p}) - D_t(\boldsymbol{\alpha}^*), \tag{3.15}$$

where $\bar{\boldsymbol{g}}^{(i)}$ is an aggregated subgradient (Line 8 of Algorithm 3.2) which lies in the convex hull of $\boldsymbol{g}^{(j)} \in \partial J(\boldsymbol{w}_t)\ \forall j \leq i$, and $\boldsymbol{\alpha}^*$ is the optimal dual solution; equations A.19–A.21 in Appendix A.1 provide intermediate steps that lead to the inequality in (3.15). Theorem A.2.3 (Appendix A.2) shows that $\epsilon^{(i)}$ is monotonically decreasing, leading us to a practical stopping criterion (Line 6 of Algorithm 3.2) for our direction-finding procedure.

A detailed derivation of Algorithm 3.2 is given in Appendix A.1, where we also prove that at a non-optimal iterate a direction-finding tolerance $\epsilon \geq 0$ exists such that the search direction produced by Algorithm 3.2 is a descent direction; in Appendix A.2 we prove that Algorithm 3.2 converges to a solution with precision $\epsilon$ in $O(1/\epsilon)$ iterations. Our proofs are based on the assumption that the spectrum (eigenvalues) of BFGS' approximation $\boldsymbol{B}_t$ to the inverse Hessian is bounded from above and below. This is a reasonable assumption if simple safeguards such as those described in Section 3.2.4 are employed in the practical implementation.

### 3.2.3 Subgradient Line Search

Given the current iterate $\boldsymbol{w}_t$ and a search direction $\boldsymbol{p}_t$, the task of a line search is to find a step size $\eta > 0$ which reduces the objective function value along the ray $\boldsymbol{w}_t + \eta\boldsymbol{p}_t$, *i.e.*, reduces the value of the one-dimensional function $\Phi(\eta)$ as defined in (2.6). Using the chain rule, we can obtain the subdifferential of $\Phi$

$$\partial\,\Phi(\eta) := \{\boldsymbol{g}^\top\boldsymbol{p}_t : \boldsymbol{g} \in \partial J(\boldsymbol{w}_t + \eta\boldsymbol{p}_t)\}. \tag{3.16}$$

Exact line search finds the optimal step size $\eta^*$ by minimizing $\Phi(\eta)$, such that $0 \in \partial\Phi(\eta^*)$; inexact line searches solve (2.6) approximately while enforcing conditions designed to ensure convergence. The original Wolfe conditions, however, require the objective function to be smooth; to extend them to nonsmooth convex problems, we propose the following subgradient reformulation:

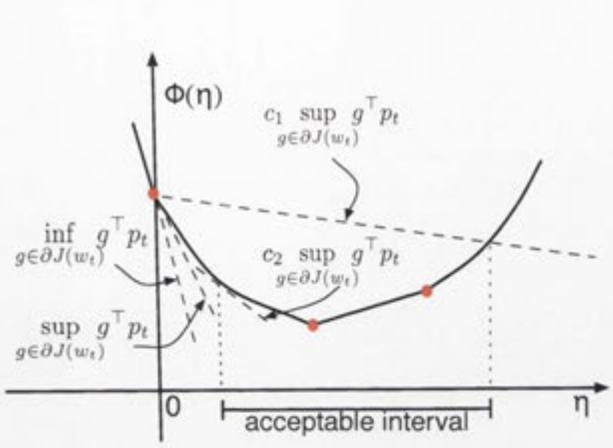
$$J(\boldsymbol{w}_{t+1}) \;\le\; J(\boldsymbol{w}_t) \;+\; c_1\eta_t \sup_{\boldsymbol{g}\in\partial J(\boldsymbol{w}_t)} \boldsymbol{g}^\top\boldsymbol{p}_t \qquad \text{(sufficient decrease)} \tag{3.17}$$

$$\text{and} \qquad \sup_{\boldsymbol{g}'\in\partial J(\boldsymbol{w}_{t+1})} \boldsymbol{g}'^\top\boldsymbol{p}_t \;\ge\; c_2 \sup_{\boldsymbol{g}\in\partial J(\boldsymbol{w}_t)} \boldsymbol{g}^\top\boldsymbol{p}_t, \qquad \text{(curvature)} \tag{3.18}$$

where $0 < c_1 < c_2 < 1$. Figure 3.6 illustrates how these conditions enforce acceptance of non-trivial step sizes that decrease the objective function value. In Appendix A.3 we formally show that for any given descent direction we can always find a positive step size that satisfies (3.17) and (3.18). Moreover, Appendix A.4 shows that the sufficient decrease condition (3.17) provides a necessary condition for the global convergence of subBFGS.

Employing an exact line search is a common strategy to speed up convergence, but it drastically increases the probability of landing on a non-differentiable point (as in Figure 3.2, left). In order to leverage the fast convergence provided by an exact line search, one must therefore use an optimizer that can handle subgradients, like our subBFGS.

Similar to the case of exact line search on smooth objective functions (*cf.* Section 2.1.2), the optimal step size $\eta^*$ obtained by an exact line search satisfies the reformulated Wolfe conditions (*resp.* the standard Wolfe conditions when $J$ is smooth) may violate the sufficient decrease condition (3.17). The curvature condition (3.18), on



$\Phi(\eta)$
$c_1 \sup_{g \in \partial J(w_t)} g^\top p_t$
$\inf_{g \in \partial J(w_t)} g^\top p_t$
$c_2 \sup_{g \in \partial J(w_t)} g^\top p_t$
$\sup_{g \in \partial J(w_t)} g^\top p_t$
0
acceptable interval
$\eta$

**Figure 3.6:** Geometric illustration of the subgradient Wolfe conditions (3.17) and (3.18). Solid disks are subdifferentiable points; the slopes of dashed lines are indicated.

the other hand, is always satisfied by $\eta^*$, as long as $\boldsymbol{p}_t$ is a descent direction (3.10):

$$\sup_{\boldsymbol{g}' \in J(\boldsymbol{w}_t + \eta^* \boldsymbol{p}_t)} \boldsymbol{g}'^\top \boldsymbol{p}_t = \sup_{g \in \partial\Phi(\eta^*)} g \geq 0 > \sup_{\boldsymbol{g} \in \partial J(\boldsymbol{w}_t)} \boldsymbol{g}^\top \boldsymbol{p}_t \tag{3.19}$$

because $0 \in \partial\,\Phi(\eta^*)$.

### 3.2.4 Bounded Spectrum of BFGS' Inverse Hessian Estimate

Recall from Section 2.1.3 that to ensure positivity of BFGS' estimate $\boldsymbol{B}_t$ of the inverse Hessian, we must have $(\forall t)\ \boldsymbol{s}_t^\top \boldsymbol{y}_t > 0$. Extending this condition to nonsmooth functions, we require

$$(\boldsymbol{w}_{t+1} - \boldsymbol{w}_t)^\top (\boldsymbol{g}_{t+1} - \boldsymbol{g}_t) > 0, \text{ where } \boldsymbol{g}_{t+1} \in \partial J(\boldsymbol{w}_{t+1}) \text{ and } \boldsymbol{g}_t \in \partial J(\boldsymbol{w}_t). \tag{3.20}$$

By Theorem 2.1.2 if $J$ is strongly convex and $\boldsymbol{w}_{t+1} \neq \boldsymbol{w}_t$, then (3.20) holds for any choice of $\boldsymbol{g}_{t+1}$ and $\boldsymbol{g}_t$.[5] For general convex functions, $\boldsymbol{g}_{t+1}$ needs to be chosen (Line 12 of Algorithm 3.1) to satisfy (3.20). The existence of such a subgradient is guaranteed by convexity of the objective function. To see this, we first use the fact that $\eta_t \boldsymbol{p}_t = \boldsymbol{w}_{t+1} - \boldsymbol{w}_t$ and $\eta_t > 0$ to rewrite (3.20) as

$$\boldsymbol{p}_t^\top \boldsymbol{g}_{t+1} > \boldsymbol{p}_t^\top \boldsymbol{g}_t, \text{ where } \boldsymbol{g}_{t+1} \in \partial J(\boldsymbol{w}_{t+1}) \text{ and } \boldsymbol{g}_t \in \partial J(\boldsymbol{w}_t). \tag{3.21}$$

It follows from (3.16) that both sides of inequality (3.21) are subgradients of $\Phi(\eta)$ at $\eta_t$ and 0, respectively. Furthermore, Theorem 3.2.1 below shows that the subdifferenital $\partial\Phi(\eta)$ is monotonically increasing with $\eta$:

[5]We found empirically that no qualitative difference between using random subgradients versus choosing a particular subgradient when updating the $\boldsymbol{B}_t$ matrix.

**Theorem 3.2.1** (Hiriart-Urruty and Lemaréchal, 1993, Theorem I.4.2.1)
*Let $\Phi$ be a one-dimensional convex function on its domain, then $\partial\Phi(\eta)$ is increasing in the sense that $g_1 \leq g_2$ whenever $g_1 \in \partial\Phi(\eta_1)$, $g_2 \in \partial\Phi(\eta_2)$, and $\eta_1 < \eta_2$.*

Therefore, $\boldsymbol{p}_t^\top \boldsymbol{g}_{t+1}$ can not be less than $\boldsymbol{p}_t^\top \boldsymbol{g}_t$ for any choice of $\boldsymbol{g}_{t+1}$ and $\boldsymbol{g}_t$, *i.e.*,

$$\inf_{\boldsymbol{g}\in\partial J(\boldsymbol{w}_{t+1})} \boldsymbol{p}_t^\top \boldsymbol{g} \geq \sup_{\boldsymbol{g}\in\partial J(\boldsymbol{w}_t)} \boldsymbol{p}_t^\top \boldsymbol{g}. \tag{3.22}$$

This means that the only case where inequality (3.21) is violated is when both terms of (3.22) are equal and in addition

$$\boldsymbol{g}_{t+1} = \operatorname*{arg\,inf}_{\boldsymbol{g}\in\partial J(\boldsymbol{w}_{t+1})} \boldsymbol{g}^\top \boldsymbol{p}_t \text{ and } \boldsymbol{g}_t = \operatorname*{arg\,sup}_{\boldsymbol{g}\in\partial J(\boldsymbol{w}_t)} \boldsymbol{g}^\top \boldsymbol{p}_t, \tag{3.23}$$

that is, in this case $\boldsymbol{p}_t^\top \boldsymbol{g}_{t+1} = \boldsymbol{p}_t^\top \boldsymbol{g}_t$. To avoid this, we simply need to set $\boldsymbol{g}_{t+1}$ to a different subgradient in $\partial J(\boldsymbol{w}_{t+1})$.

Our convergence analysis for the direction-finding procedure (Algorithm 3.2) as well as the global convergence proof of subBFGS in Appendix A.4 require the spectrum of $\boldsymbol{B}_t$ to be bounded from above and below by a positive scalar:

$$\exists\, (h, H : 0 < h \leq H < \infty) : (\forall t)\ h \preceq \boldsymbol{B}_t \preceq H. \tag{3.24}$$

From a theoretical point of view it is difficult to guarantee (3.24) (Nocedal and Wright, 1999, page 212), but based on the fact that $\boldsymbol{B}_t$ is an approximation to the inverse Hessian $\boldsymbol{H}_t^{-1}$, it is reasonable to expect (3.24) to be true if

$$(\forall t)\ 1/H \preceq \boldsymbol{H}_t \preceq 1/h. \tag{3.25}$$

Since BFGS "senses" the Hessian via (2.24) only through the parameter and gradient displacements $\boldsymbol{s}_t$ and $\boldsymbol{y}_t$, we can translate the bounds on the spectrum of $\boldsymbol{H}_t$ into conditions that only involve $\boldsymbol{s}_t$ and $\boldsymbol{y}_t$:

$$(\forall t)\ \frac{\boldsymbol{s}_t^\top \boldsymbol{y}_t}{\boldsymbol{s}_t^\top \boldsymbol{s}_t} \geq \frac{1}{H} \text{ and } \frac{\boldsymbol{y}_t^\top \boldsymbol{y}_t}{\boldsymbol{s}_t^\top \boldsymbol{y}_t} \leq \frac{1}{h}, \text{ with } 0 < h \leq H < \infty. \tag{3.26}$$

This technique is used in (Nocedal and Wright, 1999, Theorem 8.5). If $J$ is strongly convex and $\boldsymbol{s}_t \neq \boldsymbol{0}$, then by Theorem 2.1.2, there exists an $H$ such that the left inequality in (3.26) holds. On general convex functions, one can skip BFGS' curvature update if $(\boldsymbol{s}_t^\top \boldsymbol{y}_t / \boldsymbol{s}_t^\top \boldsymbol{s}_t)$ falls below a threshold. To establish the second inequality, we add a fraction of $\boldsymbol{y}_t$ to $\boldsymbol{s}_t$ at Line 14 of Algorithm 3.1 (though this modification is never actually invoked in our experiments of Chapter 4, where we set $h = 10^{-8}$).

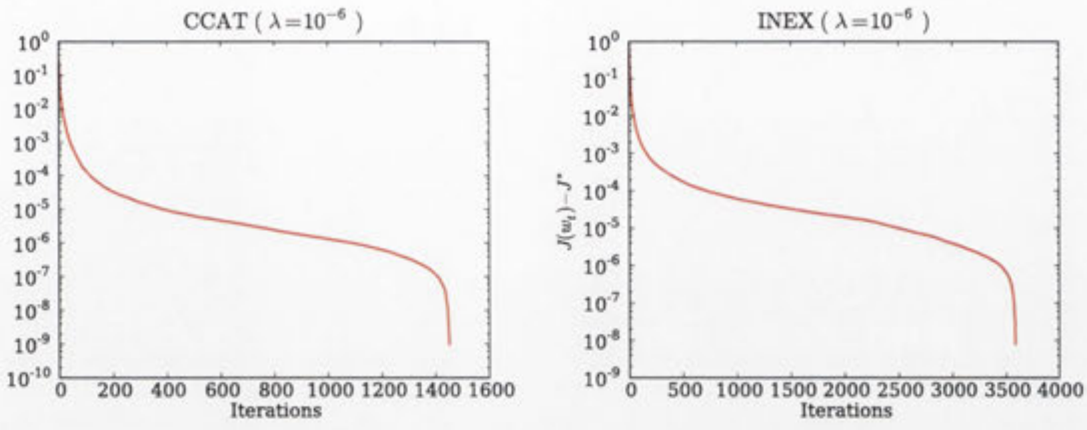


**Figure 3.7:** Convergence of subLBFGS in objective function value on sample $L_2$-regularized risk minimization problems with binary (left) and multiclass (right) hinge losses.

### 3.2.5 Limited-Memory Subgradient BFGS

It is straightforward to implement an LBFGS variant of our subBFGS algorithm: we simply modify Algorithms 3.1 and 3.2 to compute all products between $\boldsymbol{B}_t$ and a vector by means of the standard LBFGS matrix-free scheme (Algorithm 2.3). We call the resulting algorithm subLBFGS.

### 3.2.6 Convergence of Subgradient (L)BFGS

In Section 3.2.4 we have shown that the spectrum of subBFGS' inverse Hessian estimate is bounded. From this and other technical assumptions, we prove in Appendix A.4 that subBFGS is globally convergent in objective function value, *i.e.*, $J(\boldsymbol{w}) \to \inf_{\boldsymbol{w}} J(\boldsymbol{w})$. Moreover, in Appendix A.5 we show that subBFGS converges for all counterexamples we could find in the literature used to illustrate the non-convergence of existing optimization methods on nonsmooth problems.

We have also examined the convergence of subLBFGS empirically. In most of our experiments of Section 4.1, we observe that after an initial transient, subLBFGS observes a period of linear convergence, until close to the optimum it exhibits superlinear convergence behavior. This is illustrated in Figure 3.7, where we plot (on a log scale) the excess objective function value $J(\boldsymbol{w}_t)$ over its "optimum" $J^*$[6] against the iteration number in two typical runs. The same kind of convergence behavior was observed by Lewis and Overton (2008a, Figure 5.7), who applied the classical BFGS algorithm with a specially designed line search to nonsmooth functions. They caution that the apparent superlinear convergence may be an artifact caused by the inaccuracy of the

[6]Estimated empirically by running subLBFGS for $10^4$ seconds, or until the relative improvement over 5 iterations was less than $10^{-8}$.

estimated optimal value of the objective.

## 3.3 SubBFGS for $L_2$-Regularized Binary Hinge Loss

Many machine learning algorithms can be viewed as minimizing the $L_2$-regularized risk (1.1). A loss function commonly used for binary classification is the binary hinge loss (1.2). $L_2$-regularized risk minimization with the binary hinge loss is a convex but non-smooth optimization problem; in this section we show how subBFGS (Algorithm 3.1) can be applied to this problem.

Let $\mathcal{E}$, $\mathcal{M}$, and $\mathcal{W}$ index the set of points which are in error, on the margin, and well-classified, respectively:

$$\begin{aligned}
\mathcal{E} &:= \{i \in \{1, 2, \ldots, n\} : 1 - z_i \boldsymbol{w}^\top \boldsymbol{x}_i > 0\}, \\
\mathcal{M} &:= \{i \in \{1, 2, \ldots, n\} : 1 - z_i \boldsymbol{w}^\top \boldsymbol{x}_i = 0\}, \\
\mathcal{W} &:= \{i \in \{1, 2, \ldots, n\} : 1 - z_i \boldsymbol{w}^\top \boldsymbol{x}_i < 0\}.
\end{aligned}$$

Differentiating (1.1) after plugging in (1.2) then yields

$$\partial J(\boldsymbol{w}) \;=\; \lambda \boldsymbol{w} - \frac{1}{n}\sum_{i=1}^{n} \beta_i z_i \boldsymbol{x}_i \;=\; \bar{\boldsymbol{w}} - \frac{1}{n}\sum_{i\in\mathcal{M}} \beta_i z_i \boldsymbol{x}_i, \tag{3.27}$$

$$\text{where} \quad \bar{\boldsymbol{w}} := \lambda \boldsymbol{w} - \frac{1}{n}\sum_{i\in\mathcal{E}} z_i \boldsymbol{x}_i \quad \text{and} \quad \beta_i := \begin{cases} 1 & \text{if } \; i \in \mathcal{E}, \\ [0,1] & \text{if } \; i \in \mathcal{M}, \\ 0 & \text{if } \; i \in \mathcal{W}. \end{cases}$$

### 3.3.1 Efficient Oracle for the Direction-Finding Method

Recall that subBFGS requires an oracle that provides $\arg\sup_{\boldsymbol{g}\in\partial J(\boldsymbol{w}_t)} \boldsymbol{g}^\top \boldsymbol{p}$ for a given direction $\boldsymbol{p}$. For $L_2$-regularized risk minimization with the binary hinge loss we can implement such an oracle at a computational cost of $O(d\,|\,\mathcal{M}_t\,|)$, where $d$ is the dimensionality of $\boldsymbol{p}$ and $|\,\mathcal{M}_t\,|$ the number of current margin points, which is normally much less than $n$. Towards this end, we use (3.27) to obtain

$$\begin{aligned}
\sup_{\boldsymbol{g}\in\partial J(\boldsymbol{w}_t)} \boldsymbol{g}^\top \boldsymbol{p} \;&=\; \sup_{\beta_i, i\in\mathcal{M}_t} \left(\bar{\boldsymbol{w}}_t - \frac{1}{n}\sum_{i\in\mathcal{M}_t} \beta_i z_i \boldsymbol{x}_i\right)^{\!\top} \boldsymbol{p} \\
&=\; \bar{\boldsymbol{w}}_t^\top \boldsymbol{p} - \frac{1}{n}\sum_{i\in\mathcal{M}_t} \inf_{\beta_i\in[0,1]} (\beta_i z_i \boldsymbol{x}_i^\top \boldsymbol{p}).
\end{aligned} \tag{3.28}$$

**Algorithm 3.3** Exact Line Search for $L_2$-Regularized Binary Hinge Loss

```
1: input $\boldsymbol{w}, \boldsymbol{p}, \lambda, \boldsymbol{f}$, and $\Delta \boldsymbol{f}$ as in (3.30)
2: output optimal step size
3: $h = \lambda \|\boldsymbol{p}\|^2,\ j := 1$
4: $\boldsymbol{\eta} := [(\mathbf{1} - \boldsymbol{f})./\Delta \boldsymbol{f}, 0]$        (vector of subdifferentiable points & zero)
5: $\boldsymbol{\pi} = \texttt{argsort}(\boldsymbol{\eta})$        (indices sorted by non-descending value of $\boldsymbol{\eta}$)
6: while $\eta_{\pi_j} \leq 0$ do
7:     $j := j + 1$
8: end while
9: $\eta := \eta_{\pi_j}/2$
10: for $i := 1$ to $\boldsymbol{f}$.size do
11:     $\delta_i := \begin{cases} 1 & \text{if } f_i + \eta \Delta f_i < 1 \\ 0 & \text{otherwise} \end{cases}$        (value of $\boldsymbol{\delta}(\eta)$ (3.32) for any $\eta \in (0, \eta_{\pi_j})$)
12: end for
13: $\varrho := \boldsymbol{\delta}^\top \Delta \boldsymbol{f}/n - \lambda \boldsymbol{w}^\top \boldsymbol{p}$
14: $\eta := 0,\ \varrho' := 0$
15: $g := -\varrho$        (value of $\sup \partial\, \Phi(0)$)
16: while $g < 0$ do
17:     $\varrho' := \varrho$
18:     if $j > \boldsymbol{\pi}$.size then
19:         $\eta := \infty$        (no more subdifferentiable points)
20:         break
21:     else
22:         $\eta := \eta_{\pi_j}$
23:     end if
24:     repeat
25:         $\varrho := \begin{cases} \varrho - \Delta f_{\pi_j}/n & \text{if } \delta_{\pi_j} = 1 \\ \varrho + \Delta f_{\pi_j}/n & \text{otherwise} \end{cases}$        (move to next subdifferentiable point and update $\varrho$ accordingly)
26:         $j := j + 1$
27:     until $\eta_{\pi_j} \neq \eta_{\pi_{j-1}}$ and $j \leq \boldsymbol{\pi}$.size
28:     $g := \eta h - \varrho$        (value of $\sup \partial\, \Phi(\eta_{\pi_{j-1}})$)
29: end while
30: return $\min(\eta,\ \varrho'/h)$        (cf. equation 3.35)
```

Since for a given $\boldsymbol{p}$ the first term of the right-hand side of (3.28) is a constant, the supremum is attained when we set $\beta_i\ \forall i \in \mathcal{M}_t$ via the following strategy:

$$\beta_i := \begin{cases} 0 & \text{if } z_i \boldsymbol{x}_i^\top \boldsymbol{p}_t \geq 0, \\ 1 & \text{if } z_i \boldsymbol{x}_i^\top \boldsymbol{p}_t < 0. \end{cases}$$

### 3.3.2 Implementing the Line Search

The one-dimensional convex function $\Phi(\eta) := J(\boldsymbol{w} + \eta \boldsymbol{p})$ (Figure 3.8, left) obtained by restricting $J$ to a line can be evaluated efficiently. To see this, rewrite (1.1) with the hinge loss (1.2) as

$$J(\boldsymbol{w}) \ :=\ \frac{\lambda}{2}\|\boldsymbol{w}\|^2 +\ \frac{1}{n}\mathbf{1}^\top \max(\mathbf{0},\ \mathbf{1} - \boldsymbol{z} \cdot \boldsymbol{X}\boldsymbol{w}), \tag{3.29}$$

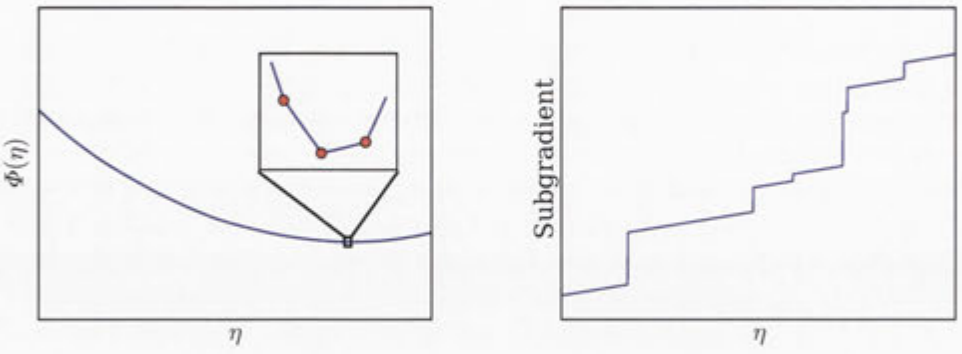


**Figure 3.8:** Left: Piecewise quadratic convex function $\Phi$ of step size $\eta$; solid disks in the zoomed inset are subdifferentiable points. Right: The subgradient of $\Phi(\eta)$ increases monotonically with $\eta$, and jumps discontinuously at subdifferentiable points.

where $\mathbf{0}$ and $\mathbf{1}$ are column vectors of zeros and ones, respectively, $\cdot$ denotes the Hadamard (component-wise) product, and $\boldsymbol{z} \in \mathbb{R}^n$ collects correct labels corresponding to each row of data in $\boldsymbol{X} := [\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_n]^\top \in \mathbb{R}^{n \times d}$. Given a search direction $\boldsymbol{p}$ at a point $\boldsymbol{w}$, (3.29) allows us to write

$$\Phi(\eta) = \frac{\lambda}{2}\|\boldsymbol{w}\|^2 + \lambda\,\eta\,\boldsymbol{w}^\top\boldsymbol{p} + \frac{\lambda\,\eta^2}{2}\|\boldsymbol{p}\|^2 + \frac{1}{n}\mathbf{1}^\top \max\left[0,\ (\mathbf{1} - (\boldsymbol{f} + \eta\,\Delta\boldsymbol{f}))\right], \quad (3.30)$$

where $\boldsymbol{f} := \boldsymbol{z} \cdot \boldsymbol{X}\boldsymbol{w}$ and $\Delta\boldsymbol{f} := \boldsymbol{z} \cdot \boldsymbol{X}\boldsymbol{p}$. Differentiating (3.30) with respect to $\eta$ gives the subdifferential of $\Phi$:

$$\partial\,\Phi(\eta) = \lambda\boldsymbol{w}^\top\boldsymbol{p} + \eta\lambda\|\boldsymbol{p}\|^2 - \frac{1}{n}\boldsymbol{\delta}(\eta)^\top\Delta\boldsymbol{f}, \quad (3.31)$$

where $\boldsymbol{\delta} : \mathbb{R} \to \mathbb{R}^n$ outputs a column vector $[\delta_1(\eta), \delta_2(\eta), \cdots, \delta_n(\eta)]^\top$ with

$$\delta_i(\eta) := \begin{cases} 1 & \text{if } f_i + \eta\,\Delta f_i < 1, \\ [0,1] & \text{if } f_i + \eta\,\Delta f_i = 1, \\ 0 & \text{if } f_i + \eta\,\Delta f_i > 1. \end{cases} \quad (3.32)$$

We cache $\boldsymbol{f}$ and $\Delta\boldsymbol{f}$, expending $O(nd)$ computational effort and using $O(n)$ storage. We also cache the scalars $\frac{\lambda}{2}\|\boldsymbol{w}\|^2$, $\lambda\,\boldsymbol{w}^\top\boldsymbol{p}$, and $\frac{\lambda}{2}\|\boldsymbol{p}\|^2$, each of which requires $O(d)$ work. The evaluation of $\mathbf{1} - (\boldsymbol{f} + \eta\,\Delta\boldsymbol{f})$, $\boldsymbol{\delta}(\eta)$, and the inner products in the final terms of (3.30) and (3.31) all take $O(n)$ effort. Given the cached terms, all other terms in (3.30) can be computed in constant time, thus reducing the cost of evaluating $\Phi(\eta)$ (*resp.* its subgradient) to $O(n)$. Furthermore, from (3.32) we see that $\Phi(\eta)$ is differentiable

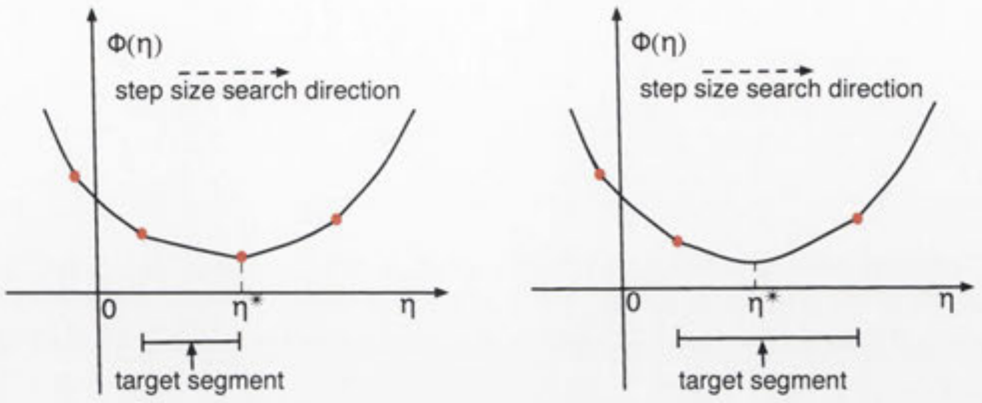


**Figure 3.9:** Nonsmooth convex function $\Phi$ of step size $\eta$. Solid disks are subdifferentiable points; the optimal step $\eta^*$ either falls on such a point (left), or lies between two such points (right).

everywhere except at

$$\eta_i := (1 - f_i)/\Delta f_i \text{ with } \Delta f_i \neq 0, \tag{3.33}$$

where it becomes subdifferentiable. At these points an element of the indicator vector (3.32) changes from 0 to 1 or vice versa (causing the subgradient to jump, as shown in Figure 3.8, right); otherwise $\boldsymbol{\delta}(\eta)$ remains constant. Using this property of $\boldsymbol{\delta}(\eta)$, we can update the last term of (3.31) in constant time when passing a hinge point (Line 25 of Algorithm 3.3). We are now in a position to introduce an exact line search which takes advantage of this scheme.

### 3.3.2.1 Exact Line Search

Given a direction $\boldsymbol{p}$, exact line search finds the optimal step size $\eta^* := \operatorname{argmin}_{\eta \geq 0} \Phi(\eta)$ that satisfies $0 \in \partial \Phi(\eta^*)$, or equivalently

$$\inf \partial \Phi(\eta^*) \leq 0 \leq \sup \partial \Phi(\eta^*). \tag{3.34}$$

By Theorem 3.2.1, $\sup \partial \Phi(\eta)$ is monotonically increasing with $\eta$. Based on this property, our algorithm first builds a list of all possible subdifferentiable points and $\eta = 0$, sorted by non-descending value of $\eta$ (Lines 4–5 of Algorithm 3.3). Then, it starts with $\eta = 0$, and walks through the sorted list until it locates the "target segment", an interval $[\eta_a, \eta_b]$ between two subdifferential points with $\sup \partial \Phi(\eta_a) \leq 0$ and $\sup \partial \Phi(\eta_b) \geq 0$. We now know that the optimal step size either coincides with $\eta_b$ (Figure 3.9, left), or lies in $(\eta_a, \eta_b)$ (Figure 3.9, right). If $\eta^*$ lies in the smooth interval $(\eta_a, \eta_b)$, then setting

(3.31) to zero gives

$$\eta^* = \frac{\boldsymbol{\delta}(\eta')^\top \Delta \boldsymbol{f}/n - \lambda\, \boldsymbol{w}^\top \boldsymbol{p}}{\lambda\, \|\boldsymbol{p}\|^2}, \quad \forall \eta' \in (\eta_a, \eta_b). \tag{3.35}$$

Otherwise, $\eta^* = \eta_b$. See Algorithm 3.3 for the detailed implementation.

## 3.4 Segmenting the Pointwise Maximum of 1-D Linear Functions

The line search of Algorithm 3.3 requires a vector $\boldsymbol{\eta}$ listing the subdifferentiable points along the line $\boldsymbol{w} + \eta \boldsymbol{p}$, and sorts it in non-descending order (Line 5). For an objective function like (1.1) whose nonsmooth component is just a sum of hinge losses (1.2), this vector is very easy to compute (*cf.* (3.33)). In order to apply our line search approach to multiclass and multilabel losses (Sections 3.5.1 and 3.5.4), however, we must solve a more general problem: we need to efficiently find the subdifferentiable points of a one-dimensional piecewise linear function $\varrho : \mathbb{R} \to \mathbb{R}$ defined to be the pointwise maximum of $r$ lines:

$$\varrho(\eta) = \max_{1 \le p \le r} (b_p + \eta\, a_p), \tag{3.36}$$

where $a_p$ and $b_p$ denote the slope and offset of the $p^{\text{th}}$ line, respectively. Clearly, $\varrho$ is convex since it is the pointwise maximum of linear functions (Boyd and Vandenberghe, 2004, Section 3.2.3), *cf.* Figure 3.10(a). The difficulty here is that although $\varrho$ consists of at most $r$ line segments bounded by at most $r-1$ subdifferentiable points, there are $r(r-1)/2$ candidates for these points, namely all intersections between any two of the $r$ lines. A naive algorithm to find the subdifferentiable points of $\varrho$ would therefore take $O(r^2)$ time. In what follows, however, we show how this can be done in just $O(r \log r)$ time. In Section 3.5 we will then use this technique (Algorithm 3.4) to perform efficient exact line search in the multiclass and multilabel settings.

We begin by specifying an interval $[L, U]$ $(0 \le L < U < \infty)$ in which to find the subdifferentiable points of $\varrho$, and set $\mathbf{y} := \boldsymbol{b} + L\boldsymbol{a}$, where $\boldsymbol{a} = [a_1, a_2, \cdots, a_r]$ and $\boldsymbol{b} = [b_1, b_2, \cdots, b_r]$. In other words, $\mathbf{y}$ contains the intersections of the $r$ lines defining $\varrho(\eta)$ with the vertical line $\eta = L$. Let $\boldsymbol{\pi}$ denote the permutation that sorts $\mathbf{y}$ in non-ascending order, *i.e.*, $p < q \implies y_{\pi_p} \ge y_{\pi_q}$, and let $\varrho^{(q)}$ be the function obtained by considering only the top $q \le r$ lines at $\eta = L$, *i.e.*, the first $q$ lines in $\boldsymbol{\pi}$:

$$\varrho^{(q)}(\eta) = \max_{1 \le p \le q} (b_{\pi_p} + \eta\, a_{\pi_p}). \tag{3.37}$$

**Algorithm 3.4** Segmenting a Pointwise Maximum of 1-D Linear Functions

```
1: input vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ of slopes and offsets
        lower bound $L$, upper bound $U$, with $0 \le L < U < \infty$
2: output sorted stack of subdifferentiable points $\boldsymbol{\eta}$
        and corresponding active line indices $\boldsymbol{\xi}$
3: $\mathbf{y} := \boldsymbol{b} + L\boldsymbol{a}$
4: $\boldsymbol{\pi} := \mathtt{argsort}(-\mathbf{y})$          (indices sorted by non-ascending value of $\mathbf{y}$)
5: $S$.push $(L, \pi_1)$          (initialize stack)
6: for $q := 2$ to y.size do
7:   while not $S$.empty do
8:     $(\eta, \xi) := S$.top
9:     $\eta' := \dfrac{b_{\pi_q} - b_\xi}{a_\xi - a_{\pi_q}}$          (intersection of two lines)
10:    if $L < \eta' \le \eta$ or $(\eta' = L$ and $a_{\pi_q} > a_\xi)$ then
11:      $S$.pop          (cf. Figure 3.10(c))
12:    else
13:      break
14:    end if
15:  end while
16:  if $L < \eta' \le U$ or $(\eta' = L$ and $a_{\pi_q} > a_\xi)$ then
17:    $S$.push $(\eta', \pi_q)$          (cf. Figure 3.10(b))
18:  end if
19: end for
20: return $S$
```

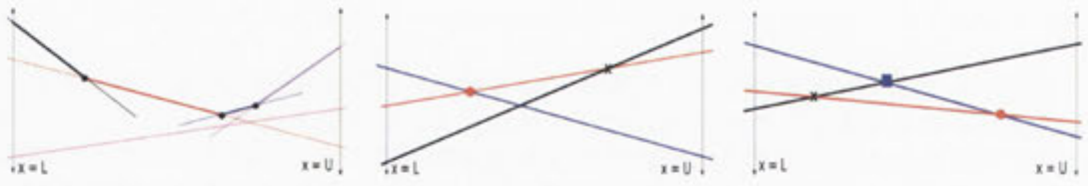
It is clear that $\varrho^{(r)} = \varrho$. Let $\boldsymbol{\eta}$ contain all $q' \le q - 1$ subdifferentiable points of $\varrho^{(q)}$ in $[L, U]$ in ascending order, and $\boldsymbol{\xi}$ the indices of the corresponding *active* lines, *i.e.*, the maximum in (3.37) is attained for line $\xi_{j-1}$ over the interval $[\eta_{j-1}, \eta_j]$: $\xi_{j-1} := \pi_{p^*}$, where $p^* = \operatorname{argmax}_{1 \le p \le q}(b_{\pi_p} + \eta a_{\pi_p})$ for $\eta \in [\eta_{j-1}, \eta_j]$, and lines $\xi_{j-1}$ and $\xi_j$ intersect at $\eta_j$.

Initially we set $\eta_0 := L$ and $\xi_0 := \pi_1$, the leftmost bold segment in Figure 3.10(a). Algorithm 3.4 goes through lines in $\boldsymbol{\pi}$ sequentially, and maintains a Last-In-First-Out stack $S$ which at the end of the $q^{\text{th}}$ iteration consists of the tuples

$$(\eta_0, \xi_0), (\eta_1, \xi_1), \ldots, (\eta_{q'}, \xi_{q'}) \tag{3.38}$$

in order of ascending $\eta_i$, with $(\eta_{q'}, \xi_{q'})$ at the top. After $r$ iterations $S$ contains a sorted list of all subdifferentiable points (and the corresponding active lines) of $\varrho = \varrho^{(r)}$ in $[L, U]$, as required by our line searches.

In iteration $q + 1$ Algorithm 3.4 examines the intersection $\eta'$ between lines $\xi_{q'}$ and $\pi_{q+1}$: If $\eta' > U$, line $\pi_{q+1}$ is irrelevant, and we proceed to the next iteration. If $\eta_{q'} < \eta' \le U$ as in Figure 3.10(b), then line $\pi_{q+1}$ is becoming active at $\eta'$, and we simply push $(\eta', \pi_{q+1})$ onto the stack. If $\eta' \le \eta_{q'}$ as in Figure 3.10(c), on the other hand, then



(a) Pointwise maximum of lines (b) Case 1 (c) Case 2

**Figure 3.10:** (a) Convex piecewise linear function defined as the maximum of 5 lines, but comprising only 4 active line segments (bold) separated by 3 subdifferentiable points (black dots). (b, c) Two cases encountered by our algorithm: (b) The new intersection (black cross) lies to the right of the previous one (red dot) and is therefore pushed onto the stack; (c) The new intersection lies to the left of the previous one. In this case the latter is popped from the stack, and a third intersection (blue square) is computed and pushed onto it.

line $\pi_{q+1}$ dominates line $\xi_{q'}$ over the interval $(\eta', \infty)$ and hence over $(\eta_{q'}, U] \subset (\eta', \infty)$, so we pop $(\eta_{q'}, \xi_{q'})$ from the stack (deactivating line $\xi_{q'}$), decrement $q'$, and repeat the comparison.

**Theorem 3.4.1** *The total running time of Algorithm 3.4 is $O(r \log r)$.*

**Proof** Computing intersections of lines as well as pushing and popping from the stack require $O(1)$ time. Each of the $r$ lines can be pushed onto and popped from the stack at most once; amortized over $r$ iterations the running time is therefore $O(r)$. The time complexity of Algorithm 3.4 is thus dominated by the initial sorting of $\mathbf{y}$ (*i.e.*, the computation of $\boldsymbol{\pi}$), which takes $O(r \log r)$ time. ■

## 3.5 SubBFGS for Multiclass and Multilabel Hinge Losses

We now use the algorithm developed in Section 3.4 to generalize the subBFGS method of Section 3.3 to the multiclass and multilabel settings with finite label set $\mathcal{Z}$. We assume that given a feature vector $\boldsymbol{x}$ our classifier predicts the label

$$z^* = \operatorname*{argmax}_{z \in \mathcal{Z}} f(\boldsymbol{w}, \boldsymbol{x}, z), \tag{3.39}$$

where $f$ is a linear function of $\boldsymbol{w}$, *i.e.*, $f(\boldsymbol{w}, \boldsymbol{x}, z) = \boldsymbol{w}^\top \phi(\boldsymbol{x}, z)$ for some feature map $\phi(\boldsymbol{x}, z)$.

### 3.5.1 Multiclass Hinge Loss

A variety of multiclass hinge losses have been proposed in the literature that generalize the binary hinge loss, and enforce a margin of separation between the true label $z_i$

and every other label. We focus on the following rather general variant (Taskar et al., 2004):[7]

$$l(\boldsymbol{x}_i, z_i, \boldsymbol{w}) \;:=\; \max_{z\in\mathcal{Z}}\,[\Delta(z, z_i) + f(\boldsymbol{w}, \boldsymbol{x}_i, z) - f(\boldsymbol{w}, \boldsymbol{x}_i, z_i)], \tag{3.40}$$

where $\Delta(z, z_i) \geq 0$ is the *label loss* specifying the margin required between labels $z$ and $z_i$. For instance, a uniform margin of separation is achieved by setting $\Delta(z, z') := \tau > 0\ \forall z \neq z'$ (Crammer and Singer, 2003a). By requiring that $\forall z \in \mathcal{Z} : \Delta(z, z) = 0$ we ensure that (3.40) always remains non-negative. Adapting (1.1) to the multiclass hinge loss (3.40) we obtain

$$J(\boldsymbol{w}) \;:=\; \frac{\lambda}{2}\|\boldsymbol{w}\|^2 + \frac{1}{n}\sum_{i=1}^{n}\max_{z\in\mathcal{Z}}\,[\Delta(z, z_i) + f(\boldsymbol{w}, \boldsymbol{x}_i, z) - f(\boldsymbol{w}, \boldsymbol{x}_i, z_i)]. \tag{3.41}$$

For a given $\boldsymbol{w}$, consider the set

$$\mathcal{Z}_i^* := \operatorname*{argmax}_{z\in\mathcal{Z}}[\Delta(z, z_i) + f(\boldsymbol{w}, \boldsymbol{x}_i, z) - f(\boldsymbol{w}, \boldsymbol{x}_i, z_i)] \tag{3.42}$$

of maximum-loss labels (possibly more than one) for the $i^{\text{th}}$ training instance. Since $f(\boldsymbol{w}, \boldsymbol{x}, z) = \boldsymbol{w}^\top\phi(\boldsymbol{x}, z)$, the subdifferential of (3.41) can then be written as

$$\partial J(\boldsymbol{w}) \;=\; \lambda\boldsymbol{w} + \frac{1}{n}\sum_{i=1}^{n}\sum_{z\in\mathcal{Z}}\beta_{i,z}\,\phi(\boldsymbol{x}_i, z) \tag{3.43}$$

$$\text{with}\quad \beta_{i,z} \;=\; \left\{\begin{array}{rl} [0,1] & \text{if } z \in \mathcal{Z}_i^* \\ 0 & \text{otherwise} \end{array}\right\} - \delta_{z,z_i}\ \text{ s.t. } \sum_{z\in\mathcal{Z}}\beta_{i,z} = 0, \tag{3.44}$$

where $\delta$ is the Kronecker delta: $\delta_{a,b} = 1$ if $a = b$, and 0 otherwise.[8]

### 3.5.2 Efficient Multiclass Direction-Finding Oracle

For $L_2$-regularized risk minimization with multiclass hinge loss, we can use a similar scheme as described in Section 3.3.1 to implement an efficient oracle that provides $\arg\sup_{\boldsymbol{g}\in\partial J(\boldsymbol{w})}\boldsymbol{g}^\top\boldsymbol{p}$ for the direction-finding procedure (Algorithm 3.2). Using (3.43),

[7]Our algorithm can also deal with the slack-rescaled variant of Tsochantaridis et al. (2005).

[8]Let $l_i^* := \max_{z\neq z_i}[\Delta(z, z_i) + f(\boldsymbol{w}, \boldsymbol{x}_i, z) - f(\boldsymbol{w}, \boldsymbol{x}_i, z_i)]$. Definition (3.44) allows the following values of $\beta_{i,z}$:

$$\left\{\begin{array}{c|ccc} & z = z_i & z \in \mathcal{Z}_i^*\backslash\{z_i\} & \text{otherwise} \\ \hline l_i^* < 0 & 0 & 0 & 0 \\ l_i^* = 0 & [-1,0] & [0,1] & 0 \\ l_i^* > 0 & -1 & [0,1] & 0 \end{array}\right\} \ \text{ s.t. } \sum_{z\in\mathcal{Z}}\beta_{i,z} = 0.$$

we can write

$$\sup_{\boldsymbol{g}\in\partial J(\boldsymbol{w})} \boldsymbol{g}^\top \boldsymbol{p} \;=\; \lambda \boldsymbol{w}^\top \boldsymbol{p} + \frac{1}{n}\sum_{i=1}^{n}\sum_{z\in\mathcal{Z}} \sup_{\beta_{i,z}} \left(\beta_{i,z}\,\phi(\boldsymbol{x}_i, z)^\top \boldsymbol{p}\right). \tag{3.45}$$

The supremum in (3.45) is attained when we pick, from the choices offered by (3.44),

$$\beta_{i,z} := \delta_{z,z_i^*} - \delta_{z,z_i}, \quad \text{where} \quad z_i^* := \operatorname*{argmax}_{z\in\mathcal{Z}_i^*} \phi(\boldsymbol{x}_i, z)^\top \boldsymbol{p}.$$

## 3.5.3 Implementing the Multiclass Line Search

Let $\Phi(\eta) := J(\boldsymbol{w} + \eta\boldsymbol{p})$ be the one-dimensional convex function obtained by restricting (3.41) to a line along direction $\boldsymbol{p}$. Letting $\varrho_i(\eta) := l(\boldsymbol{x}_i, z_i, \boldsymbol{w} + \eta\boldsymbol{p})$, we can write

$$\Phi(\eta) \;=\; \frac{\lambda}{2}\|\boldsymbol{w}\|^2 + \lambda\eta\boldsymbol{w}^\top\boldsymbol{p} + \frac{\lambda\eta^2}{2}\|\boldsymbol{p}\|^2 + \frac{1}{n}\sum_{i=1}^{n}\varrho_i(\eta). \tag{3.46}$$

Each $\varrho_i(\eta)$ is a piecewise linear convex function. To see this, observe that

$$f(\boldsymbol{w} + \eta\boldsymbol{p}, \boldsymbol{x}, z) := (\boldsymbol{w} + \eta\boldsymbol{p})^\top\phi(\boldsymbol{x}, z) = f(\boldsymbol{w}, \boldsymbol{x}, z) + \eta f(\boldsymbol{p}, \boldsymbol{x}, z) \tag{3.47}$$

and hence

$$\varrho_i(\eta) := \max_{z\in\mathcal{Z}} [\underbrace{\Delta(z, z_i) + f(\boldsymbol{w}, \boldsymbol{x}_i, z) - f(\boldsymbol{w}, \boldsymbol{x}_i, z_i)}_{=:\, b_z^{(i)}} + \eta\,\underbrace{(f(\boldsymbol{p}, \boldsymbol{x}_i, z) - f(\boldsymbol{p}, \boldsymbol{x}_i, z_i))}_{=:\, a_z^{(i)}}], \tag{3.48}$$

which has the functional form of (3.36) with $r = |\mathcal{Z}|$. Algorithm 3.4 can therefore be used to compute a sorted vector $\boldsymbol{\eta}^{(i)}$ of all subdifferentiable points of $\varrho_i(\eta)$ and corresponding active lines $\boldsymbol{\xi}^{(i)}$ in the interval $[0, \infty)$ in $O(|\mathcal{Z}|\log|\mathcal{Z}|)$ time. With some abuse of notation, we now have

$$\eta \in [\eta_j^{(i)}, \eta_{j+1}^{(i)}] \implies \varrho_i(\eta) = b_{\xi_j^{(i)}} + \eta\, a_{\xi_j^{(i)}}. \tag{3.49}$$

The first three terms of (3.46) are constant, linear, and quadratic (with non-negative coefficient) in $\eta$, respectively. The remaining sum of piecewise linear convex functions $\varrho_i(\eta)$ is also piecewise linear and convex, and so $\Phi(\eta)$ is a piecewise quadratic convex function.

**Algorithm 3.5** Exact Line Search for $L_2$-Regularized Multiclass Hinge Loss

```
1: input base point $\boldsymbol{w}$, descent direction $\boldsymbol{p}$, regularization parameter $\lambda$, vector $\boldsymbol{a}$ of
        all slopes as defined in (3.48), for each training instance $i$: sorted stack $S_i$ of
        subdifferentiable points and active lines, as produced by Algorithm 3.4
2: output optimal step size
3: $\boldsymbol{a} := \boldsymbol{a}/n,\ h := \lambda\|\boldsymbol{p}\|^2$
4: $\varrho := \lambda \boldsymbol{w}^\top \boldsymbol{p}$
5: for $i := 1$ to $n$ do
6:   while not $S_i$.empty do
7:     $R_i$.push $S_i$.pop                                  (reverse the stacks)
8:   end while
9:   $(\cdot, \xi_i) := R_i$.pop
10:  $\varrho := \varrho + a_{\xi_i}$
11: end for
12: $\eta := 0,\ \varrho' = 0$
13: $g := \varrho$                                          (value of $\sup \partial\Phi(0)$)
14: while $g < 0$ do
15:   $\varrho' := \varrho$
16:   if $\forall i: R_i$.empty then
17:     $\eta := \infty$                        (no more subdifferentiable points)
18:     break
19:   end if
20:   $\mathcal{I} := \mathrm{argmin}_{1 \le i \le n}\ \eta' : (\eta', \cdot) = R_i$.top   (find the next subdifferentiable point)
21:   $\varrho := \varrho - \sum_{i \in \mathcal{I}} a_{\xi_i}$
22:   $\Xi := \{\xi_i : (\eta, \xi_i) := R_i$.pop$,\ i \in \mathcal{I}\}$
23:   $\varrho := \varrho + \sum_{\xi_i \in \Xi} a_{\xi_i}$
24:   $g := \varrho + \eta h$                                   (value of $\sup \partial\Phi(\eta)$)
25: end while
26: return $\min(\eta, -\varrho'/h)$
```

### 3.5.3.1 Exact Multiclass Line Search

Our exact line search employs a similar two-stage strategy as discussed in Section 3.3.2.1 for locating its minimum $\eta^* := \mathrm{argmin}_{\eta>0} \Phi(\eta)$: we first find the first *subdifferentiable* point $\check{\eta}$ past the minimum, then locate $\eta^*$ within the differentiable region to its left. We precompute and cache a vector $\boldsymbol{a}^{(i)}$ of all the slopes $a_z^{(i)}$ (offsets $b_z^{(i)}$ are not needed), the subdifferentiable points $\boldsymbol{\eta}^{(i)}$ (sorted in ascending order via Algorithm 3.4), and the corresponding indices $\boldsymbol{\xi}^{(i)}$ of active lines of $\varrho_i$ for all training instances $i$, as well as $\|\boldsymbol{w}\|^2$, $\boldsymbol{w}^\top\boldsymbol{p}$, and $\lambda\|\boldsymbol{p}\|^2$.

Since $\Phi(\eta)$ is convex, any point $\eta < \eta^*$ cannot have a non-negative subgradient.[9]

[9] If $\Phi(\eta)$ has a flat optimal region, we define $\eta^*$ to be the infimum of that region.

The first subdifferentiable point $\check{\eta} \geq \eta^*$ therefore obeys

$$\begin{aligned} \check{\eta} &:= \min \eta \in \{\boldsymbol{\eta}^{(i)}, i = 1, 2, \ldots, n\} : \eta \geq \eta^* \\ &= \min \eta \in \{\boldsymbol{\eta}^{(i)}, i = 1, 2, \ldots, n\} : \sup \partial \Phi(\eta) \geq 0. \end{aligned} \tag{3.50}$$

We solve (3.50) via a simple linear search: Starting from $\eta = 0$, we walk from one subdifferentiable point to the next until $\sup \partial \Phi(\eta) \geq 0$. To perform this walk efficiently, define a vector $\boldsymbol{\psi} \in \mathbb{N}^n$ of indices into the sorted vector $\boldsymbol{\eta}^{(i)}$ *resp.* $\boldsymbol{\xi}^{(i)}$; initially $\boldsymbol{\psi} := \mathbf{0}$, indicating that $(\forall i)\ \eta_0^{(i)} = 0$. Given the current index vector $\boldsymbol{\psi}$, the next subdifferentiable point is then

$$\eta' := \eta^{(i')}_{(\psi_{i'}+1)}, \quad \text{where} \quad i' = \underset{1 \leq i \leq n}{\operatorname{argmin}}\, \eta^{(i)}_{(\psi_i+1)}; \tag{3.51}$$

the step is completed by incrementing $\psi_{i'}$, *i.e.*, $\psi_{i'} := \psi_{i'} + 1$ so as to remove $\eta^{(i')}_{\psi_{i'}}$ from future consideration.[10] Note that computing the argmin in (3.51) takes $O(\log n)$ time (*e.g.*, using a priority queue). Inserting (3.49) into (3.46) and differentiating, we find that

$$\sup \partial \Phi(\eta') \;=\; \lambda \boldsymbol{w}^\top \boldsymbol{p} + \lambda \eta' \|\boldsymbol{p}\|^2 + \frac{1}{n} \sum_{i=1}^n a_{\xi^{(i)}_{\psi_i}}. \tag{3.52}$$

The key observation here is that after the initial calculation of $\sup \partial \Phi(0) = \lambda \boldsymbol{w}^\top \boldsymbol{p} + \frac{1}{n} \sum_{i=1}^n a_{\xi_0^{(i)}}$ for $\eta = 0$, the sum in (3.52) can be updated incrementally in constant time through the addition of $a_{\xi^{(i')}_{\psi_{i'}}} - a_{\xi^{(i')}_{(\psi_{i'}-1)}}$ (Lines 20–23 of Algorithm 3.5).

Suppose we find $\check{\eta} = \eta^{(i')}_{\psi_{i'}}$ for some $i'$. We then know that the minimum $\eta^*$ is either equal to $\check{\eta}$ (Figure 3.9, left), or found within the quadratic segment immediately to its left (Figure 3.9, right). We thus decrement $\psi_{i'}$ (*i.e.*, take one step back) so as to index the segment in question, set the right-hand side of (3.52) to zero, and solve for $\eta'$ to obtain

$$\eta^* \;=\; \min\left( \check{\eta},\ \frac{\lambda \boldsymbol{w}^\top \boldsymbol{p} + \frac{1}{n} \sum_{i=1}^n a_{\xi^{(i)}_{\psi_i}}}{-\lambda \|\boldsymbol{p}\|^2} \right). \tag{3.53}$$

This only takes constant time: we have cached $\boldsymbol{w}^\top \boldsymbol{p}$ and $\lambda \|\boldsymbol{p}\|^2$, and the sum in (3.53) can be obtained incrementally by adding $a_{\xi^{(i')}_{\psi_{i'}}} - a_{\xi^{(i')}_{(\psi_{i'}+1)}}$ to its last value in (3.52).

To locate $\check{\eta}$ we have to walk at most $O(n|\mathcal{Z}|)$ steps, each requiring $O(\log n)$ computation of argmin as in (3.51). Given $\check{\eta}$, the exact minimum $\eta^*$ can be obtained in

[10]For ease of exposition, we assume $i'$ in (3.51) is unique, and deal with multiple choices of $i'$ in Algorithm 3.5.

$O(1)$. Including the preprocessing cost of $O(n|\mathcal{Z}|\log|\mathcal{Z}|)$ (for invoking Algorithm 3.4), our exact multiclass line search therefore takes $O(n|\mathcal{Z}|(\log n|\mathcal{Z}|))$ time in the worst case. Algorithm 3.5 provides an implementation which instead of an index vector $\boldsymbol{\psi}$ directly uses the sorted stacks of subdifferentiable points and active lines produced by Algorithm 3.4. (The cost of reversing those stacks in Lines 6–8 of Algorithm 3.5 can easily be avoided through the use of double-ended queues.)

### 3.5.4 Multilabel Hinge Loss

Recently, there has been interest in extending the concept of the hinge loss to multilabel problems. Multilabel problems generalize the multiclass setting in that each training instance $\boldsymbol{x}_i$ is associated with a set of labels $\mathcal{Z}_i \subseteq \mathcal{Z}$ (Crammer and Singer, 2003b). For a uniform margin of separation $\tau$, a hinge loss can be defined in this setting as follows:

$$l(\boldsymbol{x}_i, \mathcal{Z}_i, \boldsymbol{w}) := \max[0,\ \tau + \max_{z' \notin \mathcal{Z}_i} f(\boldsymbol{w}, \boldsymbol{x}_i, z') - \min_{z \in \mathcal{Z}_i} f(\boldsymbol{w}, \boldsymbol{x}_i, z)]. \tag{3.54}$$

We can generalize this to a not necessarily uniform label loss $\Delta(z', z) \geq 0$ as follows:

$$l(\boldsymbol{x}_i, \mathcal{Z}_i, \boldsymbol{w}) := \max_{\substack{(z,z'):\, z \in \mathcal{Z}_i \\ z' \notin \mathcal{Z}_i \setminus \{z\}}} [\Delta(z', z) + f(\boldsymbol{w}, \boldsymbol{x}_i, z') - f(\boldsymbol{w}, \boldsymbol{x}_i, z)], \tag{3.55}$$

where as before we require that $\Delta(z, z) = 0\ \forall z \in \mathcal{Z}$ so that by explicitly allowing $z' = z$ we can ensure that (3.55) remains non-negative. For a uniform margin $\Delta(z', z) = \tau\ \ \forall z' \neq z$ our multilabel hinge loss (3.55) reduces to the decoupled version (3.54), which in turn reduces to the multiclass hinge loss (3.40) if $\mathcal{Z}_i := \{z_i\}$ for all $i$.

For a given $\boldsymbol{w}$, let

$$\mathcal{Z}_i^* := \operatorname*{argmax}_{\substack{(z,z'):\, z \in \mathcal{Z}_i \\ z' \notin \mathcal{Z}_i \setminus \{z\}}} [\Delta(z', z) + f(\boldsymbol{w}, \boldsymbol{x}_i, z') - f(\boldsymbol{w}, \boldsymbol{x}_i, z)] \tag{3.56}$$

be the set of worst label pairs (possibly more than one) for the ith training instance. The subdifferential of the multilabel analogue of $L_2$-regularized multiclass objective (3.41) can then be written just as in (3.43), with coefficients

$$\beta_{i,z} := \sum_{z':\, (z',z) \in \mathcal{Z}_i^*} \gamma_{z',z}^{(i)} - \sum_{z':\, (z,z') \in \mathcal{Z}_i^*} \gamma_{z,z'}^{(i)}, \quad \text{where} \quad (\forall i) \sum_{(z,z') \in \mathcal{Z}_i^*} \gamma_{z,z'}^{(i)} = 1 \text{ and } \gamma_{z,z'}^{(i)} \geq 0. \tag{3.57}$$

Now let $(z_i, z_i') := \operatorname{argmax}_{(z,z') \in \mathcal{Z}_i^*} [\phi(\boldsymbol{x}_i, z') - \phi(\boldsymbol{x}_i, z)]^\top \boldsymbol{p}$ be a single steepest worst label pair in direction $\boldsymbol{p}$. We obtain $\arg\sup_{\boldsymbol{g} \in \partial J(\boldsymbol{w})} \boldsymbol{g}^\top \boldsymbol{p}$ for our direction-finding procedure by picking, from the choices offered by (3.57), $\gamma_{z,z'}^{(i)} := \delta_{z,z_i} \delta_{z',z_i'}$.

Finally, the line search we described in Section 3.5.3 for the multiclass hinge loss can be extended in a straightforward manner to our multilabel setting. The only caveat is that now $\varrho_i(\eta) := l(\boldsymbol{x}_i, \mathcal{Z}_i, \boldsymbol{w} + \eta \boldsymbol{p})$ must be written as

$$\varrho_i(\eta) := \max_{\substack{(z,z'):\, z \in \mathcal{Z}_i \\ z' \notin \mathcal{Z}_i \setminus \{z\}}} [\underbrace{\Delta(z', z) + f(\boldsymbol{w}, \boldsymbol{x}_i, z') - f(\boldsymbol{w}, \boldsymbol{x}_i, z)}_{=:\, b^{(i)}_{z,z'}} + \eta \underbrace{(f(\boldsymbol{p}, \boldsymbol{x}_i, z') - f(\boldsymbol{p}, \boldsymbol{x}_i, z))}_{=:\, a^{(i)}_{z,z'}}]. \tag{3.58}$$

In the worst case, (3.58) could be the piecewise maximum of $O(|\mathcal{Z}|^2)$ lines, thus increasing the overall complexity of the line search. In practice, however, the set of true labels $\mathcal{Z}_i$ is usually small, typically of size 2 or 3 (*cf.* Crammer and Singer, 2003b, Figure 3). As long as $\forall i : |\mathcal{Z}_i| = O(1)$, our complexity estimates of Section 3.5.3.1 still apply.

## 3.6 Related Work

We discuss related work in two areas: nonsmooth convex optimization and the problem of segmenting the pointwise maximum of a set of one-dimensional linear functions.

### 3.6.1 Nonsmooth Convex Optimization

There are three main approaches to nonsmooth convex optimization: quasi-Newton methods, bundle methods,and smooth approximation. We discuss each of these briefly, and compare and contrast our work with the state of the art.

#### 3.6.1.1 Nonsmooth Quasi-Newton Methods

These methods try to find a descent quasi-Newton direction at every iteration, and invoke a line search to minimize the one-dimensional convex function along that direction. We note that the line search routines we describe in Sections 3.3–3.5 are applicable to all such methods. An example of this class of algorithms is the work of Lukšan and Vlček (1999), who propose an extension of BFGS to nonsmooth convex problems. Their algorithm samples subgradients around non-differentiable points in order to obtain a descent direction. In many machine learning problems evaluating the objective function and its (sub)gradient is very expensive, making such an approach inefficient. In contrast, given a current iterate $\boldsymbol{w}_t$, our direction-finding routine (Algorithm 3.2) samples subgradients from the set $\partial J(\boldsymbol{w}_t)$ via the oracle. Since this avoids the cost of explicitly evaluating new (sub)gradients, it is computationally more efficient.

Recently, Andrew and Gao (2007) introduced a variant of LBFGS, the Orthant-Wise Limited-memory Quasi-Newton (OWL-QN) algorithm, suitable for optimizing

$L_1$-regularized log-linear models:

$$J(\boldsymbol{w}) := \lambda\|\boldsymbol{w}\|_1 + \underbrace{\frac{1}{n}\sum_{i=1}^{n}\ln(1+e^{-z_i\boldsymbol{w}^\top\boldsymbol{x}_i})}_{\text{logistic loss}}, \tag{3.59}$$

where the logistic loss is smooth, but the regularizer is only subdifferentiable at points where $\boldsymbol{w}$ has zero elements. From the optimization viewpoint this objective is very similar to $L_2$-regularized hinge loss; the direction finding and line search methods that we discussed in Sections 3.2.2 and 3.2.3, respectively, can be applied to this problem with slight modifications.

OWL-QN is based on the observation that the $L_1$ regularizer is linear within any given orthant. Therefore, it maintains an approximation $\boldsymbol{B}^{\text{ow}}$ to the inverse Hessian of the logistic loss, and uses an efficient scheme to select orthants for optimization. In fact, its success greatly depends on its direction-finding subroutine, which demands a specially chosen subgradient $\boldsymbol{g}^{\text{ow}}$ (Andrew and Gao, 2007, Equation 4) to produce the quasi-Newton direction, $\boldsymbol{p}^{\text{ow}} = \pi(\boldsymbol{p}, \boldsymbol{g}^{\text{ow}})$, where $\boldsymbol{p} := -\boldsymbol{B}^{\text{ow}}\boldsymbol{g}^{\text{ow}}$ and the projection $\pi$ returns a search direction by setting the $i^{\text{th}}$ element of $\boldsymbol{p}$ to zero whenever $p_i g_i^{\text{ow}} > 0$. As shown in Section 4.1.4, the direction-finding subroutine of OWL-QN can be replaced by our Algorithm 3.2, which makes OWL-QN more robust to the choice of subgradients.

#### 3.6.1.2 Bundle Methods

Bundle method solvers (Hiriart-Urruty and Lemaréchal, 1993) use past (sub)gradients to build a model of the objective function. The (sub)gradients are used to lower-bound the objective by a piecewise linear function which is minimized to obtain the next iterate. This fundamentally differs from the BFGS approach of using past gradients to approximate the (inverse) Hessian, hence building a quadratic model of the objective function.

Bundle methods have recently been adapted to the machine learning context, where they are known as SVMStruct (Tsochantaridis et al., 2005) *resp.* BMRM (Smola et al., 2007). One notable feature of these variants is that they do not employ a line search. This is justified by noting that a line search involves computing the value of the objective function multiple times, a potentially expensive operation in machine learning applications.

Franc and Sonnenburg (2008) speed up the convergence of SVMStruct for $L_2$-regularized binary hinge loss. The main idea of their optimized cutting plane algorithm, OCAS, is to perform a line search along the line connecting two successive iterates of a bundle method solver. Recently they have extended OCAS to multiclass classification (Franc and Sonnenburg, 2009). Although developed independently, their line search methods for both settings are very similar to the methods we describe in Sections 3.3.2.1 and 3.5.3.1, respectively. In particular, their line search for multiclass classification also involves segmenting the pointwise maximum of $r$ 1-D linear functions (*cf.* Section 3.4), though the $O(r^2)$ time complexity of their method is worse than our $O(r \log r)$.

#### 3.6.1.3 Smooth Approximation

Another possible way to bypass the complications caused by the nonsmoothness of an objective function is to work on a smooth approximation instead — see for instance the recent work of Nesterov (2005) and Nemirovski (2005). Some machine learning applications have also been pursued along these lines (Lee and Mangasarian, 2001; Zhang and Oles, 2001). Although this approach can be effective, it is unclear how to build a smooth approximation in general. Furthermore, smooth approximations often sacrifice dual sparsity, which often leads to better generalization performance on the test data, and also may be needed to prove generalization bounds.

### 3.6.2 Segmenting the Pointwise Maximum of 1-D Linear Functions

The problem of computing the line segments that comprise the pointwise maximum of a given set of line segments has received attention in the area of computational geometry; see Agarwal and Sharir (2000) for a survey. Hershberger (1989) for instance proposed a divide-and-conquer algorithm for this problem with the same time complexity as our Algorithm 3.4. The Hershberger (1989) algorithm solves a slightly harder problem — his function is the pointwise maximum of line segments, as opposed to our lines — but our algorithm is conceptually simpler and easier to implement.

A similar problem has also been studied under the banner of kinetic data structures by Basch (1999), who proposed a heap-based algorithm for this problem and proved a worst-case $O(r \log^2 r)$ bound, where $r$ is the number of line segments. Basch (1999) also claims that the lower bound is $O(r \log r)$; our Algorithm 3.4 achieves this bound.

## 3.7 Discussion

We proposed subBFGS (*resp.* subLBFGS), an extension of the BFGS quasi-Newton method (*resp.* its limited-memory variant) for handling nonsmooth convex optimization problems, and proved its global convergence in objective function value. We demonstrated the use of our algorithm on a variety of machine learning problems employing $L_2$-regularized binary hinge loss and its multiclass and multilabel generalizations.

Our solver is easy to parallelize: The master node computes the search direction and transmits it to the slaves. The slaves compute the (sub)gradient and loss value on subsets of data. This information is then aggregated at the master node, and used to compute the next search direction before the process repeats. Similarly, the line search, which is the expensive part of the computation on multiclass and multilabel problems, is easy to parallelize: the slaves run Algorithm 3.4 on subsets of the data; the results are fed back to the master who can then run Algorithm 3.5 to compute the step size.

Our algorithms rely on an efficient exact line search. We proposed such line searches for the binary hinge loss and its generalizations to the multiclass and multilabel settings. The exact line searches for the multiclass and multilabel hinge losses are based on a conceptually simple yet optimal algorithm to segment the pointwise maximum of lines. A crucial assumption we had to make is that the number $|\mathcal{Z}|$ of labels is manageable since it takes $O(|\mathcal{Z}| \log |\mathcal{Z}|)$ time to identify the hinges associated with each training instance. In certain structured prediction problems (Tsochantaridis et al., 2005) which have recently gained prominence in machine learning, the set $\mathcal{Z}$ could be exponentially large — for instance, predicting binary labels on a chain of length $n$ produces $2^n$ possible labels. Clearly, our line searches are not efficient in such cases.

Finally, to put our contributions in perspective, recall that we modified three aspects of the standard BFGS algorithm, namely the quadratic model (Section 3.2.1), the descent direction finding (Section 3.2.2), and the Wolfe conditions (Section 3.2.3). Each of these modifications is versatile enough to be used as a component in other nonsmooth optimization algorithms. This not only offers the promise of improving existing algorithms, but may also help clarify connections between them. We hope that our research will focus attention on the core subroutines that need to be made more efficient in order to handle larger and larger datasets.

Chapter 4

# SubLBFGS for Nonsmooth Convex Optimization

In this chapter we compare the performance of our limited-memory subBFGS (subLBFGS, Section 3.2.5 ) algorithm with other state-of-the-art nonsmooth optimization methods on $L_2$-regularized binary, multiclass, and multilabel hinge loss minimization problems. We also compare the specialized $L_1$-regularized logistic loss optimizer OWL-QN (Section 3.6.1) with a variant that uses our direction-finding routine (Algorithm 3.2). In all these contexts our methods perform comparable to or better than specialized state-of-the-art solvers on a number of publicly available datasets. Open source software implementing our algorithms is available for download.

## 4.1 Experiments

In all experiments the regularization parameter was chosen from the set $10^{\{-6,-5,\cdots,-1\}}$ so as to achieve the highest prediction accuracy on the test dataset, while convergence behavior (objective function value *vs.* CPU seconds) is reported on the training dataset. To see the influence of the regularization parameter $\lambda$, we also compared the time required by each algorithm to reduce the objective function value to within 2% of the optimal value.[1] For all algorithms the initial iterate $\boldsymbol{w}_0$ was set to $\mathbf{0}$. Open source C++ code implementing our algorithms and experiments is available for download from `http://www.cs.adelaide.edu.au/~jinyu/Code/nonsmoothOpt.tar.gz`

The subgradient for the construction of the subLBFGS search direction (*cf.* Line 12 of Algorithm 3.1) was chosen arbitrarily from the subdifferential. For the binary hinge loss minimization (Section 4.1.3), for instance, we picked an arbitrary subgradient by randomly setting the coefficient $\beta_i \; \forall i \in \mathcal{M}$ in (3.27) to either 0 or 1.

[1]For $L_1$-regularized logistic loss minimization, the "optimal" value was the final objective value achieved by the OWL-QN* algorithm (*cf.* Section 4.1.4). In all other experiments, it was found by running subLBFGS for $10^4$ seconds, or until its relative improvement over 5 iterations was less than $10^{-8}$.

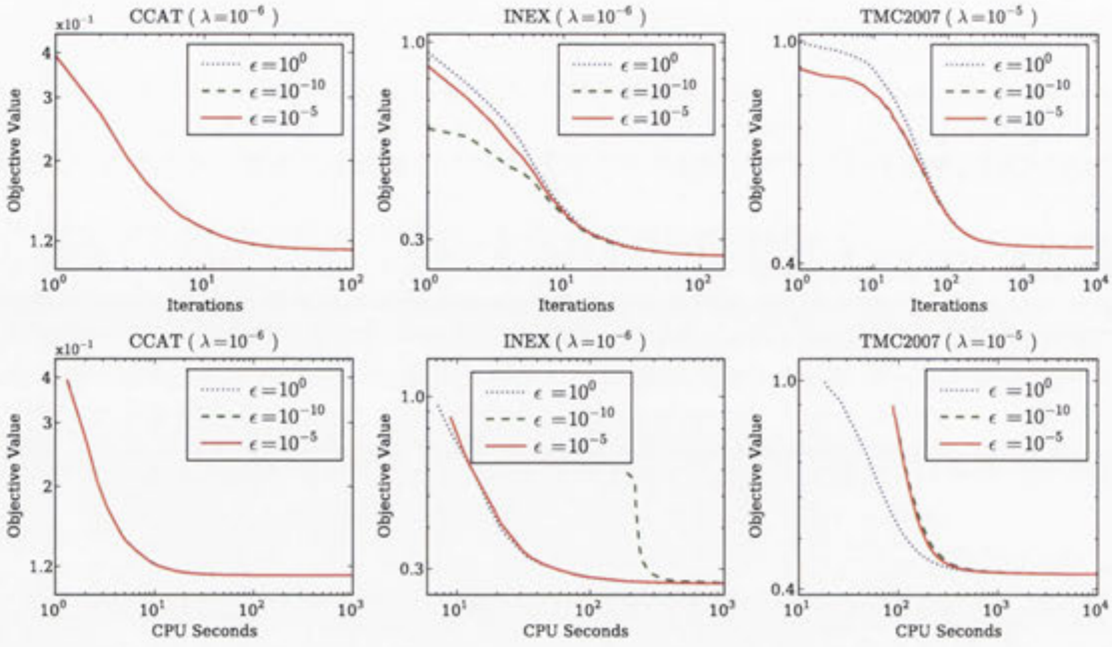


**Figure 4.1:** Performance of subLBFGS with varying direction-finding tolerance $\epsilon$ in terms of objective function value *vs.* the number of iterations (top row) *resp.* CPU seconds (bottom row) on sample $L_2$-regularized risk minimization problems with the binary (left), multiclass (center), and multilabel (right) hinge losses.

On strictly convex problems such as what we consider in this chapter, every convergent optimizer will reach the same solution; comparing generalisation performance is therefore pointless. Hence we concentrate on empirically evaluating the convergence behavior (objective function value *vs.* CPU seconds). All experiments were carried out on a Linux machine with dual 2.4 GHz Intel Core 2 processors and 4 GB of RAM.

### 4.1.1 Convergence Tolerance of the Direction-Finding Procedure

The convergence tolerance $\epsilon$ of Algorithm 3.2 controls the precision of the solution to the direction-finding problem (3.5): lower tolerance may yield a better search direction. Figure 4.1 (left) shows that on binary classification problems, subLBFGS is not sensitive to the choice of $\epsilon$ (*i.e.*, the quality of the search direction). This is due to the fact that $\partial J(\boldsymbol{w})$ as defined in (3.27) is usually dominated by its constant component $\bar{\boldsymbol{w}}$; search directions that correspond to different choices of $\epsilon$ therefore can not differ too much from each other. In the case of multiclass and multilabel classification, where the structure of $\partial J(\boldsymbol{w})$ is more complicated, we can see from Figure 4.1 (top center and right) that a better search direction can lead to faster convergence in terms of iteration numbers. However, this is achieved at the cost of more CPU time spent in the direction-finding routine. As shown in Figure 4.1 (bottom center and right), extensively optimizing

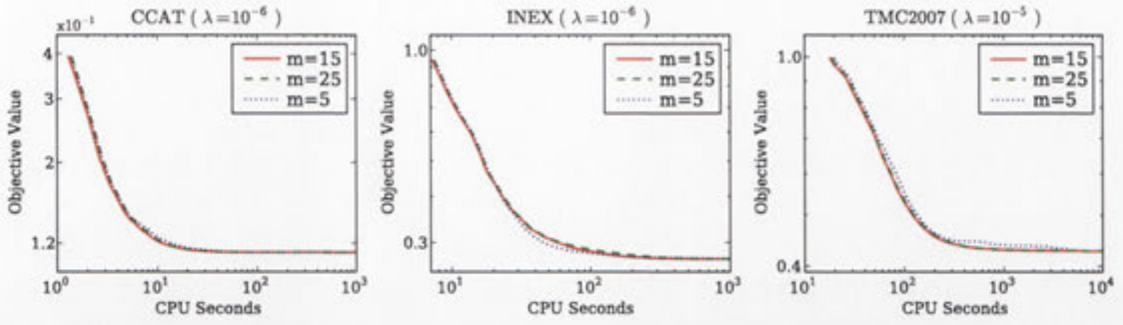


**Figure 4.2:** Performance of subLBFGS with varying buffer size on sample $L_2$-regularized risk minimization problems with the binary (left), multiclass (center), and multilabel hinge losses (right).

**Table 4.1**: The binary datasets used in our experiments of Sections 3.1, 4.1.3, and 4.1.4.

| Dataset | Train/Test Set Size | Dimensionality | Sparsity |
|---|---|---|---|
| Covertype | 522911/58101 | 54 | 77.8% |
| CCAT | 781265/23149 | 47236 | 99.8% |
| Astro-physics | 29882/32487 | 99757 | 99.9% |
| MNIST-binary | 60000/10000 | 780 | 80.8% |
| Adult9 | 32561/16281 | 123 | 88.7% |
| Real-sim | 57763/14438 | 20958 | 99.8% |
| Leukemia | 38/34 | 7129 | 00.0% |

the search direction actually slows down convergence in terms of CPU seconds. We therefore used an intermediate value of $\epsilon = 10^{-5}$ for all our experiments, except that for multiclass and multilabel classification problems we relaxed the tolerance to 1.0 at the initial iterate $\boldsymbol{w} = \boldsymbol{0}$, where the direction-finding oracle $\arg\sup_{\boldsymbol{g}\in\partial J(\boldsymbol{0})} \boldsymbol{g}^\top \boldsymbol{p}$ is expensive to compute, due to the large number of extreme points in $\partial J(\boldsymbol{0})$.

### 4.1.2 Size of SubLBFGS Buffer

The size $m$ of the subLBFGS buffer determines the number of parameter and gradient displacement vectors $\boldsymbol{s}_t$ and $\boldsymbol{y}_t$ used in the construction of the quasi-Newton direction. Figure 4.2 shows that the performance of subLBFGS is not sensitive to the particular value of $m$ within the range $5 \leq m \leq 25$. We therefore simply set $m = 15$ *a priori* for all subsequent experiments; this is a typical value for LBFGS (Nocedal and Wright, 1999).

**Table 4.2:** Regularization parameter $\lambda$ and overall number $k$ of direction-finding iterations in our experiments of Sections 4.1.3 and 4.1.4, respectively.

| | $L_1$-reg. logistic loss | | | $L_2$-reg. binary loss | |
|---|---|---|---|---|---|
| Dataset | $\lambda_{L_1}$ | $k_{L_1}$ | $k_{L_1r}$ | $\lambda_{L_2}$ | $k_{L_2}$ |
| Covertype | $10^{-5}$ | 1 | 2 | $10^{-6}$ | 0 |
| CCAT | $10^{-6}$ | 284 | 406 | $10^{-6}$ | 0 |
| Astro-physics | $10^{-5}$ | 1702 | 1902 | $10^{-4}$ | 0 |
| MNIST-binary | $10^{-4}$ | 55 | 77 | $10^{-6}$ | 0 |
| Adult9 | $10^{-4}$ | 2 | 6 | $10^{-5}$ | 1 |
| Real-sim | $10^{-6}$ | 1017 | 1274 | $10^{-5}$ | 1 |

**Table 4.3:** The multiclass (top 6 rows) and multilabel (bottom 3 rows) datasets used, values of the regularization parameter, and overall number $k$ of direction-finding iterations in our multiclass and multilabel hinge loss experiments of Section 4.1.5.

| Dataset | Train/Test Set Size | Dimensionality | $\lvert\mathcal{Z}\rvert$ | Sparsity | $\lambda$ | $k$ |
|---|---|---|---|---|---|---|
| Letter | 16000/4000 | 16 | 26 | 0.0% | $10^{-6}$ | 65 |
| USPS | 7291/2007 | 256 | 10 | 3.3% | $10^{-3}$ | 14 |
| Protein | 14895/6621 | 357 | 3 | 70.7% | $10^{-2}$ | 1 |
| MNIST | 60000/10000 | 780 | 10 | 80.8% | $10^{-3}$ | 1 |
| INEX | 6053/6054 | 167295 | 18 | 99.5% | $10^{-6}$ | 5 |
| News20 | 15935/3993 | 62061 | 20 | 99.9% | $10^{-2}$ | 12 |
| Scene | 1211/1196 | 294 | 6 | 0.0% | $10^{-1}$ | 14 |
| TMC2007 | 21519/7077 | 30438 | 22 | 99.7% | $10^{-5}$ | 19 |
| RCV1 | 21149/2000 | 47236 | 103 | 99.8% | $10^{-5}$ | 4 |

### 4.1.3 $L_2$-Regularized Binary Hinge Loss

For our first set of experiments, we applied subLBFGS with exact line search (Algorithm 3.3) to the task of $L_2$-regularized binary hinge loss minimization. Our control methods are the bundle method solver BMRM (Teo et al., 2010) and the optimized cutting plane algorithm OCAS (Franc and Sonnenburg, 2008),[2] both of which were shown to perform competitively on this task. SVMStruct (Tsochantaridis et al., 2005) is another well-known bundle method solver that is widely used in the machine learning community. For $L_2$-regularized optimization problems BMRM is identical to SVMStruct, hence we omit comparisons with SVMStruct.

Table 4.1 lists the six datasets we used: The Covertype dataset of Blackard, Jock

[2]The source code of OCAS (version 0.6.0) was obtained from `http://www.shogun-toolbox.org`.

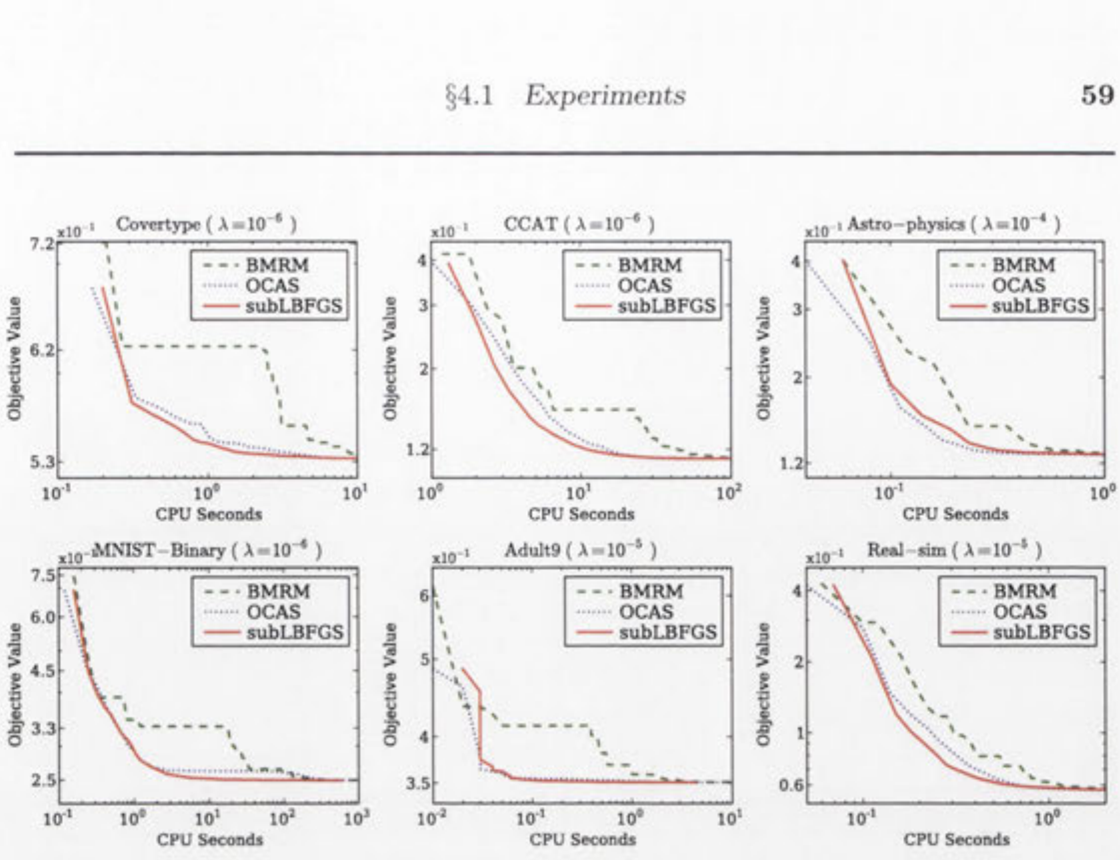


**Figure 4.3:** Objective function value *vs.* CPU seconds on $L_2$-regularized binary hinge loss minimization tasks.

& Dean,[3] CCAT from the Reuters RCV1 collection,[4] the Astro-physics dataset of abstracts of scientific papers from the Physics ArXiv (Joachims, 2006), the MNIST dataset of handwritten digits[5] with two classes: even and odd digits, the Adult9 dataset of census income data,[6] and the Real-sim dataset of real *vs.* simulated data.[6] Table 4.2 lists our parameter settings, and reports the overall number $k_{L_2}$ of iterations through the direction-finding loop (Lines 6–13 of Algorithm 3.2) for each dataset. The very small values of $k_{L_2}$ indicate that on these problems subLBFGS only rarely needs to correct its initial guess of a descent direction.

It can be seen from Figure 4.3 that subLBFGS (solid) reduces the objective value considerably faster than BMRM (dashed). On the binary MNIST dataset, for instance, the objective function value of subLBFGS after 10 CPU seconds is 25% lower than that of BMRM. In this set of experiments the performance of subLBFGS and OCAS (dotted) is very similar.

Figure 4.4 shows that all algorithms generally converge faster for larger values of the regularization constant λ. However, in most cases subLBFGS converges faster than BMRM across a wide range of λ values, exhibiting a speedup of up to more than two orders of magnitude. SubLBFGS and OCAS show similar performance here: for small values of λ, OCAS converges slightly faster than subLBFGS on the Astro-physics and

[3]`http://kdd.ics.uci.edu/databases/covertype/covertype.html`

[4]`http://www.daviddlewis.com/resources/testcollections/rcv1`

[5]`http://yann.lecun.com/exdb/mnist`

[6]`http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html`

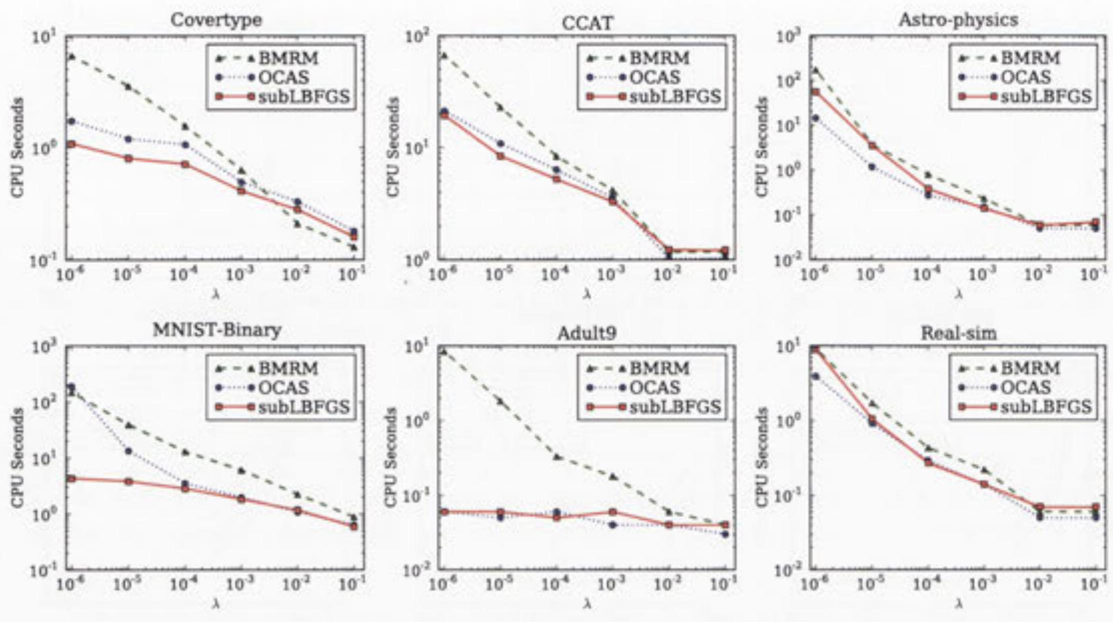


**Figure 4.4:** Regularization parameter $\lambda \in \{10^{-6}, \cdots, 10^{-1}\}$ *vs.* CPU seconds taken to reduce the objective function to within 2% of the optimal value.

Real-sim datasets but is outperformed by subLBFGS on the Covertype, CCAT, and binary MNIST datasets.

### 4.1.4 $L_1$-Regularized Logistic Loss

To demonstrate the utility of our direction-finding routine (Algorithm 3.2) in its own right, we plugged it into the OWL-QN algorithm (Andrew and Gao, 2007)[7] as an alternative direction-finding method such that $\boldsymbol{p}^{\text{ow}} = \texttt{descentDirection}(\boldsymbol{g}^{\text{ow}}, \epsilon, k_{\max})$, and compared this variant (denoted OWL-QN*) with the original (*cf.* Section 3.6.1) on $L_1$-regularized minimization of the logistic loss (3.59), on the same datasets as in Section 4.1.3.

An oracle that supplies $\arg\sup_{\boldsymbol{g}\in\partial J(\boldsymbol{w})} \boldsymbol{g}^\top \boldsymbol{p}$ for this objective is easily constructed by noting that (3.59) is nonsmooth whenever at least one component of the parameter vector $\boldsymbol{w}$ is zero. Let $w_i = 0$ be such a component; the corresponding component of the subdifferential $\partial\,\lambda\|\boldsymbol{w}\|_1$ of the $L_1$ regularizer is the interval $[-\lambda, \lambda]$. The supremum of $\boldsymbol{g}^\top \boldsymbol{p}$ is attained at the interval boundary whose sign matches that of the corresponding component of the direction vector $\boldsymbol{p}$, *i.e.*, at $\lambda\,\text{sign}(p_i)$.

Using the stopping criterion suggested by Andrew and Gao (2007), we ran experiments until the averaged relative change in the objective value over the previous 5 iterations fell below $10^{-5}$. As shown in Figure 4.5, the only clear difference in convergence

[7]The source code of OWL-QN (original release) was obtained from Microsoft Research through `http://tinyurl.com/p774cx`.

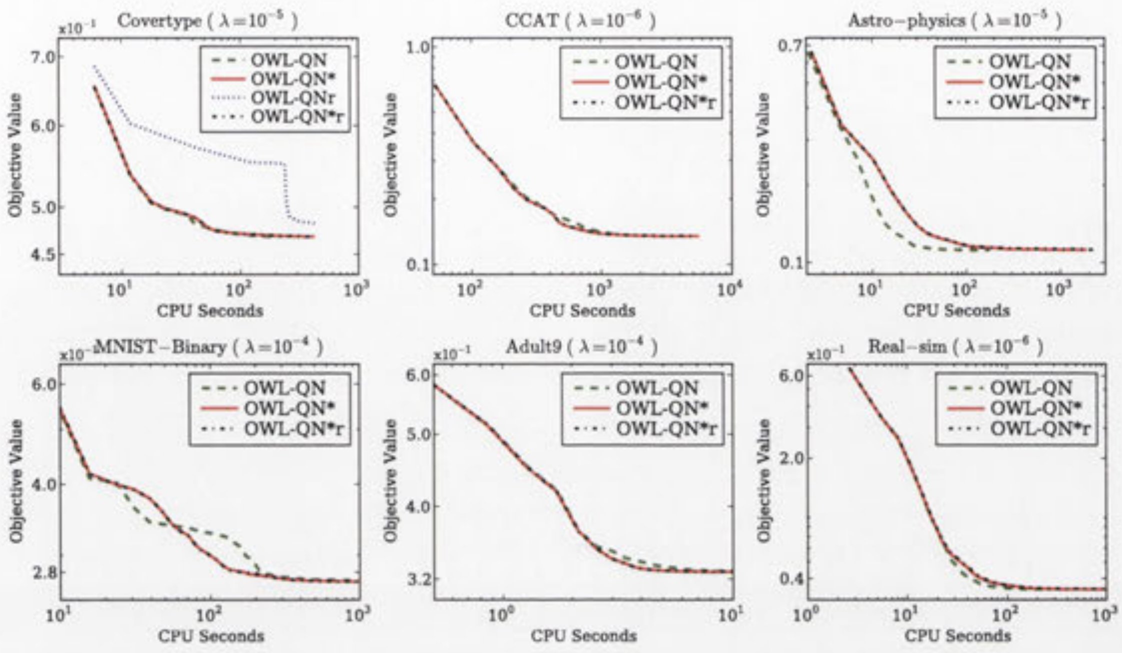


**Figure 4.5:** Objective function value *vs.* CPU seconds on $L_1$-regularized logistic loss minimization tasks.

between the two algorithms is found on the Astro-physics dataset where OWL-QN* is outperformed by the original OWL-QN method. This is because finding a descent direction via Algorithm 3.2 is particularly difficult on the Astro-physics dataset (as indicated by the large inner loop iteration number $k_{L_1}$ in Table 4.2); the slowdown on this dataset can also be found in Figure 4.6 for other values of $\lambda$. Although finding a descent direction can be challenging for the generic direction-finding routine of OWL-QN*, in the following experiment we show that this routine is very robust to the choice of initial subgradients.

To examine the algorithms' sensitivity to the choice of subgradients, we also ran them with subgradients randomly chosen from the set $\partial J(\boldsymbol{w})$ (as opposed to the specially chosen subgradient $\boldsymbol{g}^{\text{ow}}$ used in the previous set of experiments) fed to their corresponding direction-finding routines. OWL-QN relies heavily on its particular choice of subgradients, hence breaks down completely under these conditions: the only dataset where we could even plot its (poor) performance was Covertype (dotted "OWL-QNr" line in Figure 4.5). Our direction-finding routine, by contrast, is self-correcting and thus not affected by this manipulation: the curves for OWL-QN*r lie on top of those for OWL-QN*. Table 4.2 shows that in this case more direction-finding iterations are needed though: $k_{L_1r} > k_{L_1}$. This empirically confirms that as long as $\arg\sup_{\boldsymbol{g}\in\partial J(\boldsymbol{w})} \boldsymbol{g}^\top \boldsymbol{p}$ is given, Algorithm 3.2 can indeed be used as a generic quasi-Newton direction-finding routine that is able to recover from a poor initial choice

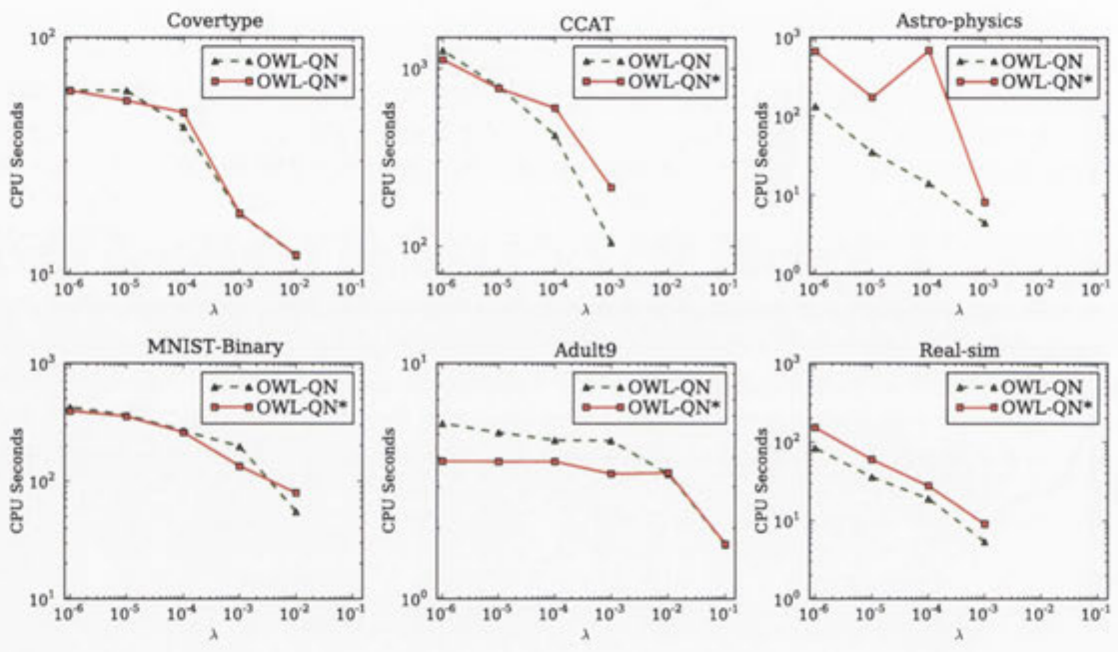


**Figure 4.6:** Regularization parameter $\lambda \in \{10^{-6}, \cdots, 10^{-1}\}$ *vs.* CPU seconds taken to reduce the objective function to within 2% of the optimal value. (No point is plotted if the initial parameter $\boldsymbol{w}_0 = \mathbf{0}$ is already optimal.)

of subgradients.

### 4.1.5 $L_2$-Regularized Multiclass and Multilabel Hinge Loss

We incorporated our exact line search of Section 3.5.3.1 into both subLBFGS and OCAS (Franc and Sonnenburg, 2008), thus enabling them to deal with multiclass and multilabel losses. We refer to our generalized version of OCAS as line search BMRM (ls-BMRM). Using the variant of the multiclass and multilabel hinge loss which enforces a uniform margin of separation ($\Delta(z, z') = 1 \ \forall z \neq z'$), we experimentally evaluated both algorithms on a number of publicly available datasets (Table 4.3). All multiclass datasets except INEX were downloaded from `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multiclass.html`, while the multilabel datasets were obtained from `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multilabel.html`. INEX is available for download from `http://webia.lip6.fr/~bordes/mywiki/doku.php?id=multiclass_data` (details can be found in Maes et al. (2007)). The original RCV1 dataset consists of 23149 training instances, of which we used 21149 instances for training and the remaining 2000 for testing.

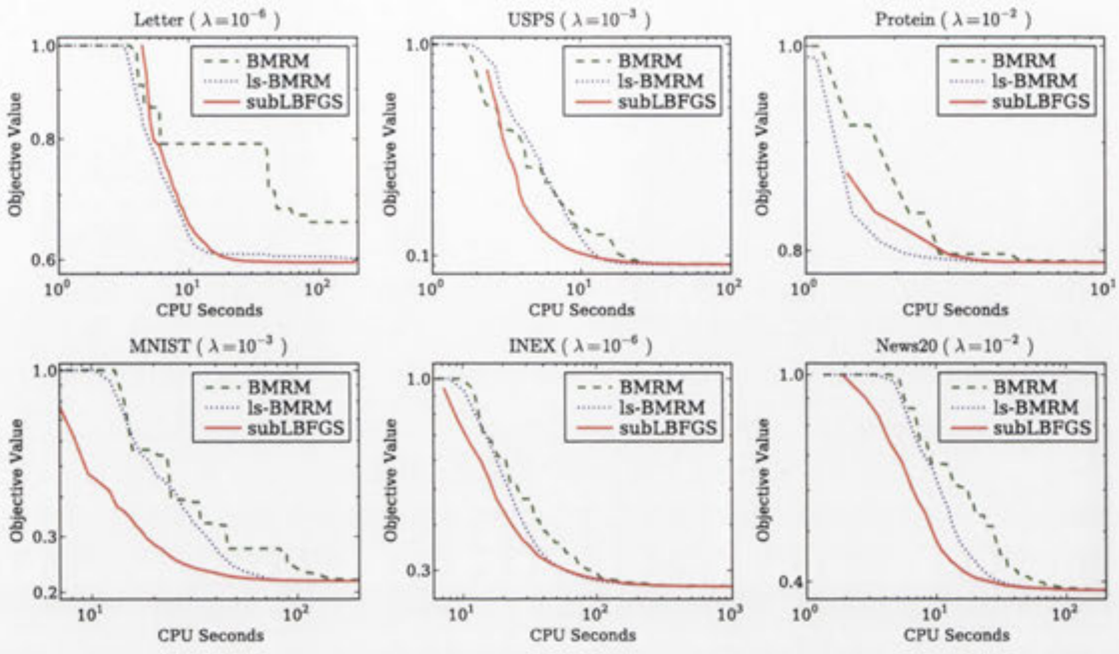


**Figure 4.7:** Objective function value *vs.* CPU seconds on $L_2$-regularized multiclass hinge loss minimization tasks.

#### 4.1.5.1 Performance on Multiclass Problems

This set of experiments is designed to demonstrate the convergence properties of multiclass subLBFGS, compared to the BMRM bundle method (Teo et al., 2010) and ls-BMRM.

Figure 4.7 shows that subLBFGS comprehensively outperforms BMRM in all cases. On the Letter dataset, the objective function value of subLBFGS after 20 CPU seconds is 24% lower than that of BMRM. On 4 out of 6 datasets, subLBFGS outperforms ls-BMRM early on but slows down later, for an overall performance comparable to ls-BMRM. On the MNIST dataset, for instance, subLBFGS takes only about half as much CPU time as ls-BMRM to reduce the objective function value to 0.3 (about 50% above the optimal value), yet both algorithms reach within 2% of the optimal value at about the same time (*cf.* Figure 4.8, bottom left). We hypothesize that subLBFGS' local model (3.4) of the objective function facilitates rapid early improvement but is less appropriate for final convergence to the optimu. Bundle methods, on the other hand, are slower initially because they need to accumulate a sufficient number of gradients to build a faithful piecewise linear model of the objective function. These results suggest that a hybrid approach that first runs subLBFGS then switches to ls-BMRM may be promising.

Similar to what we saw in the binary setting (Figure 4.4), Figure 4.8 shows that all

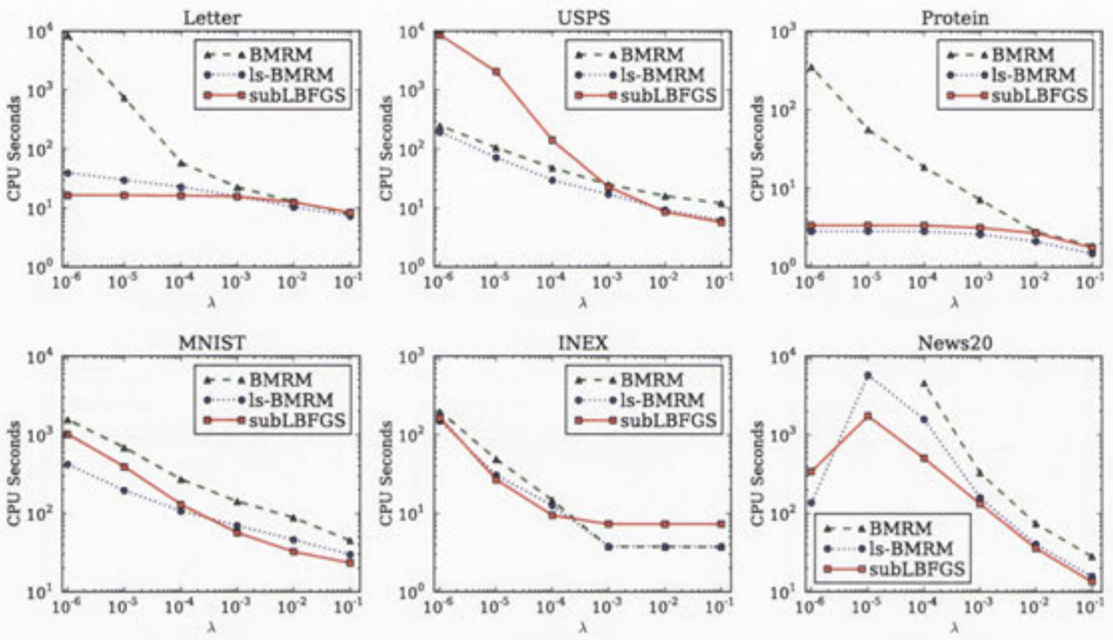


**Figure 4.8:** Regularization parameter $\lambda \in \{10^{-6}, \cdots, 10^{-1}\}$ *vs.* CPU seconds taken to reduce the objective function to within 2% of the optimal value. (No point is plotted if an algorithm fails to reach the threshold value within $10^4$ seconds.)

algorithms tend to converge faster for large values of $\lambda$. Generally, subLBFGS converges faster than BMRM across a wide range of $\lambda$ values; for small values of $\lambda$ it can greatly outperform BMRM (as seen on Letter, Protein, and News20). The performance of subLBFGS is worse than that of BMRM in two instances: on USPS for small values of $\lambda$, and on INEX for large values of $\lambda$. The poor performance on USPS may be caused by a limitation of subLBFGS' local model (3.4) that causes it to slow down on final convergence. On the INEX dataset, the initial point $\boldsymbol{w}_0 = \mathbf{0}$ is nearly optimal for large values of $\lambda$; in this situation there is no advantage in using subLBFGS.

Leveraging its exact line search (Algorithm 3.5), ls-BMRM is competitive on all datasets and across all $\lambda$ values, exhibiting performance comparable to subLBFGS in many cases. From Figure 4.8 we find that BMRM never outperforms both subLBFGS or ls-BMRM.

### 4.1.5.2 Performance on Multilabel Problems

In our final set of experiments we switch to the multilabel setting. Figure 4.9 shows that on the Scene dataset the performance of subLBFGS is similar to that of BMRM, while on the larger TMC2007 and RCV1 sets, subLBFGS outperforms both of its competitors initially but slows down later on, resulting in performance no better than

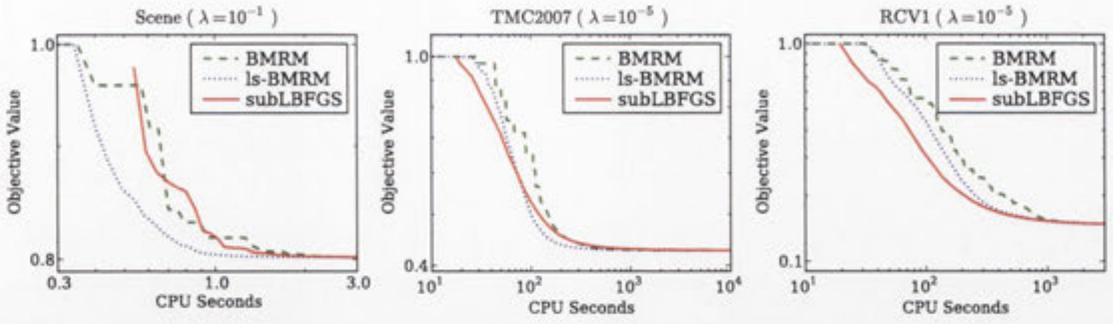


**Figure 4.9:** Objective function value *vs.* CPU seconds in $L_2$-regularized multilabel hinge loss minimization tasks.

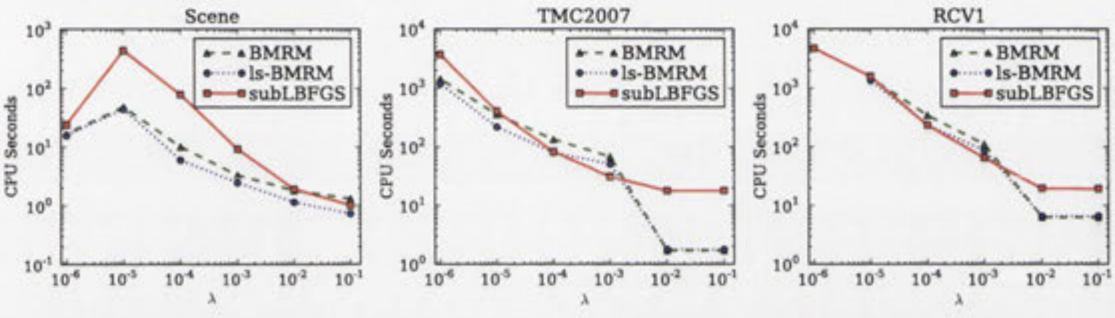


**Figure 4.10:** Regularization parameter $\lambda \in \{10^{-6}, \cdots, 10^{-1}\}$ *vs.* CPU seconds taken to reduce the objective function to within 2% of the optimal value. (No point is plotted if an algorithm fails to reach the threshold value within $10^4$ seconds.)

BMRM. Comparing performance across different values of $\lambda$ (Figure 4.10), we find that in many cases subLBFGS requires more time than its competitors to reach within 2% of the optimal value, and in contrast to the multiclass setting, here ls-BMRM only performs marginally better than BMRM. The primary reason for this is that the exact line search used by ls-BMRM and subLBFGS requires substantially more computational effort in the multilabel than in the multiclass setting. There is an inherent trade-off here: subLBFGS and ls-BMRM expend computation in an exact line search, while BMRM focuses on improving its local model of the objective function instead. In situations where the line search is very expensive, the latter strategy seems to work similarly well.

## 4.2 Discussion

We applied subLBFGS to a variety of machine learning problems employing $L_2$-regularized binary hinge loss and its multiclass and multilabel generalizations as well as $L_1$-regularized risk minimization with the logistic loss. Our experiments show that our algorithm is versatile, applicable to many problems, and often outperforms specialized solvers.

In many of our experiments we observe that subLBFGS decreases the objective function rapidly at the beginning but slows down closer to the optimum. We hypothesize that this is due to an averaging effect: Initially (*i.e.*, when sampled sparsely at a coarse scale) a superposition of many hinges looks sufficiently similar to a smooth function for optimization of a quadratic local model to work well (*cf.* Figure 3.4). Later on, when the objective is sampled at finer resolution near the optimum, the few nearest hinges begin to dominate the picture, making a smooth local model less appropriate.

Even though the local model (3.4) of sub(L)BFGS is nonsmooth, it only explicitly models the hinges at its present location — all others are subject to smooth quadratic approximation. Apparently this strategy works sufficiently well during early iterations to provide rapid improvement on multiclass problems, which typically comprise a large number of hinges. The exact location of the optimum, however, may depend on individual nearby hinges which are not represented in (3.4), resulting in the observed slowdown.

Bundle method solvers, by contrast, exhibit slow initial progress but tend to be competitive asymptotically. This is because they build a piecewise linear lower bound of the objective function, which initially is not very good but through successive tightening eventually becomes a faithful model. To take advantage of this we are contemplating hybrid solvers that switch over from sub(L)BFGS to a bundle method as appropriate.

While bundle methods like BMRM have an exact, implementable stopping criterion based on the duality gap, no such stopping criterion exists for BFGS and other quasi-Newton algorithms. Therefore, it is customary to use the relative change in function value as an implementable stopping criterion. Developing a stopping criterion for sub(L)BFGS based on duality arguments remains an important open question.

Chapter 5

# A Stochastic Quasi-Newton Method for Online Convex Optimization

In this chapter we develop variants of the BFGS quasi-Newton method, in both its full and memory-limited forms, for stochastic (online) optimization of convex functions. We begin by providing background material on stochastic gradient-based learning. A brief review of past stochastic gradient methods is given in Section 5.2. We modify BFGS so as to make it amenable to stochastic approximation (Section 5.3), before applying analogous modifications to LBFGS in Section 5.4. We then set up two stochastic quadratic problems (Section 5.5), on which we illustrate the merits of our algorithms in Section 5.6. Further experimental studies are carried out in Chapter 6, where we apply oLBFGS to the training of Condition Random Fields in natural language processing.

For ease of exposition, in this chapter we assume the objective function $J : \mathbb{R}^d \to \mathbb{R}$ to be convex and differentiable everywhere, though it has been noted (Bottou, 1998; LeCun et al., 1998) that stochastic methods are inherently robust to non-convexity and non-differentiability of an objective function.

## 5.1 Stochastic Gradient-Based Learning

In machine learning the objective function usually involves summation of loss terms over a set of training data. Classical optimization methods must compute this sum in its entirety for every parameter update. Such "batch" methods are therefore very inefficient for real-world applications involving large datasets.

Stochastic gradient methods address this problem by using gradient estimates obtained from small subsamples of the data. For instance, under the regularized risk minimization framework (1.1), the *stochastic approximation* of the regularized risk $J(\boldsymbol{w})$

only involves summation of loss terms over a *mini-batch* $\mathcal{X}$ of the training data:

$$J(\boldsymbol{w}, \mathcal{X}) := \lambda\Omega(\boldsymbol{w}) + \frac{1}{b} \sum_{(\boldsymbol{x}_i, z_i) \in \mathcal{X}} l(\boldsymbol{x}_i, z_i, \boldsymbol{w}), \tag{5.1}$$

where $\mathcal{X}$ contains a batch of $b$ pairs $(\boldsymbol{x}_i, z_i)$ of feature vectors and the corresponding labels. Usually $b$ is much smaller than the size $n$ of the training dataset. The cost of evaluating the stochastic function (and hence gradient) is therefore often far less than in the batch setting. To distinguish $J(\boldsymbol{w})$ from its data-dependent counterpart $J(\boldsymbol{w}, \mathcal{X})$, we refer to $J(\boldsymbol{w})$ as the *deterministic* objective function.

For purposes of convergence analysis, training instances in the mini-batch $\mathcal{X}$ are commonly assumed to be drawn independently according to some underlying distribution (Bottou, 1998). In practice, however, they are better chosen by repeated exhaustive sampling without replacement, implemented by repeating the following two-step procedure:

1. randomly permute the training data;

2. sequentially take batches of data until the training set is exhausted.

The simplest stochastic gradient method is stochastic gradient descent (SGD), which adjusts the parameter vector $\boldsymbol{w}$ in the direction of the negative *stochastic gradient* $-\nabla_{\boldsymbol{w}} J(\boldsymbol{w}, \mathcal{X})$. Stochastic gradient methods can be slow in converging to the optimum of the deterministic objective function (Bottou and Murata, 2002), due to the noise in the stochastic approximation of gradients. Nevertheless, it has been shown (LeCun et al., 1998) that they often can quickly obtain an approximate solution in the vicinity of the optimum, which is sufficiently accurate to ensure good generalization performance on the test dataset. Therefore, in terms of generalization performance, SGD is found to routinely outperform sophisticated batch optimization methods often by orders of magnitude on large datasets (Bottou and LeCun, 2004; Vishwanathan et al., 2006). However, it suffers from slow convergence on problems that are ill-conditioned, *i.e.*, have eigenvalues of widely differing magnitude (see *e.g.*, Bray et al., 2004, Figure 5). It is known from batch optimization that incorporating second-order information into the parameter update can greatly accelerate convergence on ill-conditioned problems, as evidenced by conjugate gradient (CG) (Shewchuk, 1994) and BFGS. A natural question to ask is whether the good asymptotic convergence of such second-order batch methods could be carried over to the stochastic setting. Schraudolph and Graepel (2003), however, show that online implementations of CG methods are ineffective; in our experiments BFGS is found to fail catastrophically on stochastic problems. The failure of these online implementations of batch methods is mainly due to the fact that core tools of conventional gradient-based optimization such as line searches and

Krylov subspaces collapse in the presence of sampling noise in stochastic approximation (Schraudolph and Graepel, 2003).

Here we overcome these limitations by modifying BFGS and LBFGS so as to obtain fast stochastic quasi-Newton methods for online convex optimization. Moreover, our modifications to (L)BFGS offer the promise of devising new stochastic methods that are able to incorporate curvature information of an objective function into the parameter update; for instance, the recently developed SGD-QN (Bordes et al., 2009) algorithm is of this kind.

## 5.2 Existing Stochastic Gradient Methods

We review three stochastic gradient optimization algorithms that are representative of the spectrum of such methods developed to date.

### 5.2.1 Stochastic Gradient Descent

Simple stochastic gradient descent (SGD) takes the form

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta_t \nabla_{\boldsymbol{w}} J(\boldsymbol{w}_t, \mathcal{X}_t), \tag{5.2}$$

where $\boldsymbol{w}_t \in \mathbb{R}^d$ is the current parameter vector, $\eta_t > 0$ a step size, and $\mathcal{X}_t$ the current mini-batch of training data. Robbins and Monro (1951) have shown that (5.2) converges to $\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} J(\boldsymbol{w})$ as $t \to \infty$, provided that the step size satisfies

$$\sum_t \eta_t = \infty \quad \text{and} \quad \sum_t \eta_t^2 < \infty\,. \tag{5.3}$$

A commonly used decay schedule for $\eta_t$ that fulfills these conditions is given by

$$\eta_t = \frac{\tau}{\tau + t}\,\eta_0\,, \tag{5.4}$$

where $\eta_0, \tau > 0$ are tuning parameters.

The SGD parameter update (5.2) takes only $O(d)$ space and time per iteration. Although it can greatly outperform sophisticated batch methods on large datasets, it suffers from slow convergence on ill-conditioned problems.

### 5.2.2 Stochastic Meta-Descent

Stochastic Meta-Descent (SMD) (Schraudolph, 1999, 2002) accelerates SGD by providing each element of the parameter vector $\boldsymbol{w}$ with its own step size:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \boldsymbol{\eta}_t \cdot \nabla J_{\boldsymbol{w}}(\boldsymbol{w}_t, \mathcal{X}_t), \tag{5.5}$$

where $\cdot$ denotes Hadamard (component-wise) multiplication. The step size vector $\boldsymbol{\eta}_t \in \mathbb{R}^d$ is adapted via a simultaneous stochastic gradient descent in log-space:

$$\begin{aligned}\ln \boldsymbol{\eta}_{t+1} &= \ln \boldsymbol{\eta}_t - \mu \sum_{i=0}^{t} \lambda^i \, \nabla_{\ln \boldsymbol{\eta}_{t-i}} J(\boldsymbol{w}_{t+1}, \mathcal{X}_{t+1}) \\ &= \ln \boldsymbol{\eta}_t - \mu \nabla_{\boldsymbol{w}_{t+1}} J(\boldsymbol{w}_{t+1}, \mathcal{X}_{t+1}) \sum_{i=0}^{t} \lambda^i \, \nabla_{\ln \boldsymbol{\eta}_{t-i}} \boldsymbol{w}_{t+1} \\ &= \ln \boldsymbol{\eta}_t - \mu \nabla_{\boldsymbol{w}_{t+1}} J(\boldsymbol{w}_{t+1}, \mathcal{X}_{t+1}) \cdot \boldsymbol{v}_{t+1}, \end{aligned} \tag{5.6}$$

where $\mu \geq 0$ is a scalar tuning parameter, and $\boldsymbol{v}_{t+1} := \sum_{i=0}^{t} \lambda^i \, \nabla_{\ln \boldsymbol{\eta}_{t-i}} \boldsymbol{w}_{t+1}$ models the dependence of the current parameter on past step sizes over a time scale governed by the decay factor $\lambda \in [0, 1]$. Elementwise exponentiation of (5.6) followed by the linearization $\exp(u) \approx \max(\frac{1}{2}, 1 + u)$ gives the desired multiplicative update

$$\begin{aligned}\boldsymbol{\eta}_{t+1} &= \boldsymbol{\eta}_t \cdot \exp(-\mu \nabla_{\boldsymbol{w}_{t+1}} J(\boldsymbol{w}_{t+1}, \mathcal{X}_{t+1}) \cdot \boldsymbol{v}_{t+1}) \\ &\approx \boldsymbol{\eta}_t \cdot \max\left(\tfrac{1}{2}, 1 - \mu \nabla_{\boldsymbol{w}_{t+1}} J(\boldsymbol{w}_{t+1}, \mathcal{X}_{t+1}) \cdot \boldsymbol{v}_{t+1}\right). \end{aligned} \tag{5.7}$$

The auxiliary vector $\boldsymbol{v}_{t+1}$ is maintained incrementally via

$$\boldsymbol{v}_{t+1} = \lambda \boldsymbol{v}_t - \boldsymbol{\eta}_t \cdot (\nabla_{\boldsymbol{w}} J(\boldsymbol{w}_t, \mathcal{X}_t) + \lambda \boldsymbol{H}_t \boldsymbol{v}_t), \tag{5.8}$$

with $\boldsymbol{H}_t := \nabla^2_{\boldsymbol{w}} J(\boldsymbol{w}_t, \mathcal{X}_t)$, *i.e.*, the *instantaneous Hessian* at time $t$, and $\boldsymbol{v}_0 = \mathbf{0}$; a detailed derivation is given by Schraudolph et al. (2006). Since $\boldsymbol{H}_t \boldsymbol{v}_t$ can be computed very efficiently (Schraudolph, 2002), SMD still takes only $O(d)$ space and time per iteration. It improves upon SGD by providing an adaptive step size decay, and handling some (but not all) forms of ill-conditioning. In the experiments of Section 5.6, however, its performance essentially equals that of SGD.

### 5.2.3 Natural Gradient Descent

The natural gradient (NG) algorithm (Amari et al., 1998) incorporates the Riemannian metric tensor $\boldsymbol{G}_t := \mathbb{E}_{\mathcal{X}}[\nabla_{\boldsymbol{w}} J(\boldsymbol{w}_t, \mathcal{X}) \nabla_{\boldsymbol{w}} J(\boldsymbol{w}_t, \mathcal{X})^\top]$ into the stochastic gradient update:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta_t \bar{\boldsymbol{G}}_t^{-1} \nabla_{\boldsymbol{w}} J(\boldsymbol{w}_t, \mathcal{X}_t), \tag{5.9}$$

with step sizes $\eta_t$ typically set by (5.4), and $\bar{\boldsymbol{G}}_t$ an estimate of $\boldsymbol{G}_t$ updated via

$$\bar{\boldsymbol{G}}_{t+1} = (1 - \frac{1}{t}) \, \bar{\boldsymbol{G}}_t + \frac{1}{t} \nabla_{\boldsymbol{w}} J(\boldsymbol{w}_t, \mathcal{X}_t) \nabla_{\boldsymbol{w}} J(\boldsymbol{w}_t, \mathcal{X}_t)^\top. \tag{5.10}$$

**Algorithm 5.1** Online BFGS Method

```
1: input
     • stochastic approximation of convex objective $J$ and its gradient $\nabla J$ over data sequence $\mathcal{X}_t$ for $t = 0, 1, 2, \cdots$
     • initial parameter vector $\boldsymbol{w}_0$
     • sequence of step sizes $\eta_t > 0$, e.g., obtained from (5.4)
     • parameters $0 < c \leq 1,\ \lambda \geq 0,\ \epsilon > 0$
2: output $\boldsymbol{w}_t$
3: $t := 0$
4: $\boldsymbol{B}_0 = \epsilon \boldsymbol{I}$
5: while not converged do
6:    $\boldsymbol{p}_t = -\boldsymbol{B}_t \nabla_{\boldsymbol{w}} J(\boldsymbol{w}_t, \mathcal{X}_t)$
7:    (no line search)
8:    $\boldsymbol{s}_t = \frac{\eta_t}{c} \boldsymbol{p}_t$
9:    $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \boldsymbol{s}_t$
10:   $\boldsymbol{y}_t = \nabla_{\boldsymbol{w}} J(\boldsymbol{w}_{t+1}, \mathcal{X}_t) - \nabla_{\boldsymbol{w}} J(\boldsymbol{w}_t, \mathcal{X}_t) + \lambda \boldsymbol{s}_t$
11:   if $t = 0$ then
12:      $\boldsymbol{B}_t := \frac{\boldsymbol{s}_t^\top \boldsymbol{y}_t}{\boldsymbol{y}_t^\top \boldsymbol{y}_t} \boldsymbol{I}$
13:   end if
14:   $\rho_t = (\boldsymbol{s}_t^\top \boldsymbol{y}_t)^{-1}$
15:   $\boldsymbol{B}_{t+1} = (\boldsymbol{I} - \rho_t \boldsymbol{s}_t \boldsymbol{y}_t^\top) \boldsymbol{B}_t (\boldsymbol{I} - \rho_t \boldsymbol{y}_t \boldsymbol{s}_t^\top) + c \rho_t \boldsymbol{s}_t \boldsymbol{s}_t^\top$
16:   $t := t + 1$
17: end while
18: return $\boldsymbol{w}_t$
```

The Sherman-Morrison formula can be employed to directly update $\bar{\boldsymbol{G}}_t^{-1}$:

$$\begin{aligned}\bar{\boldsymbol{G}}_{t+1}^{-1} &= \frac{t}{t-1}\left[\boldsymbol{G}_t + \left(\frac{1}{(t-1)}\nabla_{\boldsymbol{w}} J(\boldsymbol{w}_t, \mathcal{X}_t)\right)\nabla_{\boldsymbol{w}} J(\boldsymbol{w}_t, \mathcal{X}_t)^\top\right]^{-1} \\ &= \frac{t}{t-1}\left[\bar{\boldsymbol{G}}_t^{-1} - \frac{\bar{\boldsymbol{G}}_t^{-1}\nabla_{\boldsymbol{w}} J(\boldsymbol{w}_t, \mathcal{X}_t)\nabla_{\boldsymbol{w}} J(\boldsymbol{w}_t, \mathcal{X}_t)^\top \bar{\boldsymbol{G}}_t^{-1}}{(t-1) + \nabla_{\boldsymbol{w}} J(\boldsymbol{w}_t, \mathcal{X}_t)^\top \bar{\boldsymbol{G}}_t^{-1} \nabla_{\boldsymbol{w}} J(\boldsymbol{w}_t, \mathcal{X}_t)}\right],\end{aligned} \tag{5.11}$$

reducing the computational cost of NG from $O(d^3)$ (for inverting $\bar{\boldsymbol{G}}_t$) to $O(d^2)$ space and time per iteration — still prohibitively expensive for large $d$. Where it is affordable, NG greatly benefits from the incorporation of second-order information. Note that if the denominator of the fraction in the square brackets in (5.11) is zero, then NG's Hessian approximation $\bar{\boldsymbol{G}}_{t+1}$ is not invertible, *i.e.*, it is singular in such a case.

## 5.3 Online BFGS Method

Algorithm 5.1 shows our online BFGS (oBFGS) method, with all modifications relative to standard BFGS (Algorithm 2.1) underlined. The changes required to get BFGS to work well with stochastic approximation fall into three aspects which we shall elaborate on in turn: making do without line search, modifying the update of BFGS' inverse Hessian approximation, and taking consistent gradient measurements.

### 5.3.1 Convergence without Line Search

Line searches (Section 2.1.2) are highly problematic in the stochastic setting since the global validity of the criteria they employ such as the Wolfe conditions (2.7, 2.8) cannot be established from local subsamples of the problem.

Unlike conjugate gradient (Shewchuk, 1994), however, BFGS does not require an exact line search to correctly update its inverse Hessian estimate $\boldsymbol{B}_t$: we can actually replace the line search with a step size decay schedule such as (5.4) that satisfies (5.3) with no undue effect, provided that we can ensure $(\forall t)\ \boldsymbol{B}_t \succ 0$ by other means. For now we do this by restricting our attention to convex optimization problems (no negative eigenvalues of the Hessian $\nabla^2 J(\boldsymbol{w})$), for which

$$(\forall t) \quad \boldsymbol{s}_t^\top \boldsymbol{y}_t \ \geq \ 0 \tag{5.12}$$

holds, where $\boldsymbol{s}_t$ and $\boldsymbol{y}_t$ defined in (2.15) are the parameter *resp.* gradient displacement vectors. Zero and small eigenvalues ($\boldsymbol{s}_t^\top \boldsymbol{y}_t = 0$ and $\boldsymbol{s}_t^\top \boldsymbol{y}_t \approx 0$) are dealt with by modifying the BFGS update to estimate the inverse of $\nabla^2 J(\boldsymbol{w}) + \lambda \boldsymbol{I}$, where $\lambda \geq 0$ is a model-trust region parameter. This can be achieved by simply adding $\lambda \boldsymbol{s}_t$ to $\boldsymbol{y}_t$ at Line 10 of Algorithm 2.1. To see this, recall from Section 2.1.3 that the secant equation $\boldsymbol{B}_{t+1}\boldsymbol{y}_t = \boldsymbol{s}_t$ essentially models the following property of $\nabla^2 J(\boldsymbol{w}_t)$:

$$\nabla^2 J(\boldsymbol{w}_t)\ \boldsymbol{s}_t \ \approx \ \boldsymbol{y}_t. \tag{5.13}$$

Adding $\lambda \boldsymbol{I}$ to $\nabla^2 J(\boldsymbol{w}_t)$ in (5.13), we obtain

$$\left(\nabla^2 J(\boldsymbol{w}_t) + \lambda \boldsymbol{I}\right)\ \boldsymbol{s}_t \ \approx \ \boldsymbol{y}_t + \lambda \boldsymbol{s}_t, \tag{5.14}$$

leading to the adjustment of $\boldsymbol{y}_t$ at Line 10. Another possible way to ensure the positivity of $(\boldsymbol{s}_t^\top \boldsymbol{y}_t)$ is to adopt a strategy similar to the dampened BFGS update (Nocedal and Wright, 1999, Procedure 18.2) by setting

$$\boldsymbol{y}_t = \alpha \left[\nabla_{\boldsymbol{w}} J(\boldsymbol{w}_{t+1}, \mathcal{X}_t) - \nabla_{\boldsymbol{w}} J(\boldsymbol{w}_t, \mathcal{X}_t)\right] + (1-\alpha)\boldsymbol{B}_t^{-1}\boldsymbol{s}_t, \tag{5.15}$$

where the convex combination coefficient $\alpha \in [0,1]$ is chosen heuristically. Compared to this update, our modification to the calculation of $\boldsymbol{y}_t$ is computationally more efficient since it does not need to maintain the approximation $\boldsymbol{B}_t^{-1}$ to the Hessian.

A recent study by Sunehag et al. (2009, Theorem 3.2) shows that under some technical conditions on $\nabla^2 J(\boldsymbol{w})$ and $\nabla_{\boldsymbol{w}} J(\boldsymbol{w}, \mathcal{X})$, a second-order stochastic method converges to the optimum of a twice-differentiable convex objective function almost surely, provided that (1) the step size $\eta_t$ obeys the Robbins-Monro conditions (5.3) and (2) the spectrum of the scaling matrix is bounded from below and above by positive scalars. Our choice of step size decay schedule for oBFGS satisfies the condition on $\eta_t$. The model-trust region parameter $\lambda$ effectively provides an upper bound on the spectrum of $\boldsymbol{B}_t$; to establish a lower bound, we can simply add a fraction of $\boldsymbol{I}$ to $\boldsymbol{B}_t$, *i.e.*, add a fraction of $-\nabla_{\boldsymbol{w}} J(\boldsymbol{w}_t, \mathcal{X}_t)$ to $\boldsymbol{p}_t$ at Line 6 of Algorithm 5.1, though in the experiments of Section 5.6 we do not find it necessary to invoke this modification.

Finally, without line search we need to explicitly ensure that the first parameter update (before $\boldsymbol{B}_0$ has been appropriately scaled at Line 12) does not cause any problems. This is done by multiplying $\boldsymbol{B}_0$ at Line 4 with a very small $\epsilon > 0$ so that the first parameter update is likewise small. The value of $\epsilon$ is application-dependent but non-critical; we typically use $\epsilon = 10^{-10}$.

### 5.3.2 Modified BFGS Curvature Update

We have found empirically that scaling down the last term $\boldsymbol{s}_t \boldsymbol{s}_t^\top$ of the curvature update by a factor $c \in (0,1]$ (Line 15 of Algorithm 5.1) substantially improves the performance of oBFGS for small batch sizes. In a sense the curvature matrix $\boldsymbol{B}_{t+1}$ obtained from the modified update approximates a *dampened inverse Hessian* $c\left[\nabla^2 J(\boldsymbol{w}_t)\right]^{-1}$: using (5.13), we can write

$$c\left[\nabla^2 J(\boldsymbol{w}_t)\right]^{-1} \boldsymbol{y}_t \;\approx\; c\boldsymbol{s}_t; \tag{5.16}$$

replacing $\boldsymbol{s}_t$ in the original BFGS curvature update (Lines 12–13 of Algorithm 2.1) with $c\boldsymbol{s}_t$ gives the modified curvature update. This scaling strategy for $\boldsymbol{B}_t$ is known from standard BFGS (Brodlie, 1977).[1] We compensate for the resulting scaling of $\boldsymbol{B}_t$ by dividing the step size $\eta_t$ by $c$ at Line 8 of Algorithm 5.1. It may be possible to determine the optimal value for $c$ analytically; in the experiments reported here we simply used $c = 0.1$ throughout.

[1] Brodlie (1977, Equation 3.2) scales BFGS' Hessian estimate, instead of the inverse Hessian estimate as shown in Line 15 of Algorithm 5.1.

### 5.3.3 Consistent Gradient Measurements

We also need to account for the fact that in the stochastic setting our gradient measurements are noisy. This means that a simple convergence test like $\|\nabla J(\boldsymbol{w}_t)\| > \epsilon$ in Algorithm 2.1 must be replaced by a more robust one, for instance, checking whether the norm of a running average of recent stochastic gradients has remained below a given threshold for the last $k$ iterations.

Finally, and most importantly, care must be taken in the computation of $\boldsymbol{y}_t$ at Line 10 of Algorithm 5.1. A naive translation of the "difference of last two gradients" into the stochastic setting would compute

$$\nabla_{\boldsymbol{w}} J(\boldsymbol{w}_{t+1}, \mathcal{X}_{t+1}) - \nabla_{\boldsymbol{w}} J(\boldsymbol{w}_t, \mathcal{X}_t). \tag{5.17}$$

This would allow sampling noise to enter the BFGS' curvature update since the two terms in (5.17) are computed on different data samples.

Instead we must compute the difference of gradients on the *same* data sample $\mathcal{X}_t$ used to compute the quasi-Newton direction $\boldsymbol{p}_t$, and hence the step $\boldsymbol{s}_t$, at Lines 6 and 8, respectively.

## 5.4 Limited-Memory Online BFGS

It is straightforward to implement a limited-memory variant of our oBFGS algorithm: we simply modify the standard LBFGS (Algorithm 2.2) as follows:

- use stochastic gradients in place of deterministic gradients throughout, while always taking consistent gradient measurements as in Line 10 of Algorithm 5.1;
- modify the convergence test $\|\nabla J(\boldsymbol{w}_t)\| > \epsilon$ as discussed in Section 5.3.3;
- replace Line 5 of Algorithm 2.2 with a step size decay schedule such as (5.4) which obeys the Robbins-Monro conditions (5.3).

We also replace Line 9 of the LBFGS direction update (Algorithm 2.3) with

$$\boldsymbol{p}_t := \begin{cases} \epsilon\, \boldsymbol{p}_t & \text{if } t = 0; \\ \dfrac{\boldsymbol{p}_t}{\min(t, m)} \displaystyle\sum_{i=1}^{\min(t,m)} \dfrac{\boldsymbol{s}_{t-i}^\top \boldsymbol{y}_{t-i}}{\boldsymbol{y}_{t-i}^\top \boldsymbol{y}_{t-i}} & \text{otherwise.} \end{cases} \tag{5.18}$$

This ensures that the first parameter update is small (*cf.* Line 4 of Algorithm 5.1), and improves online performance by averaging away some of the sampling noise. Note that the oBFGS' curvature scaling factor $c \in (0, 1]$ (*cf.* Section 5.3.2) is not required here, meaning one less tuning parameter for oLBFGS.

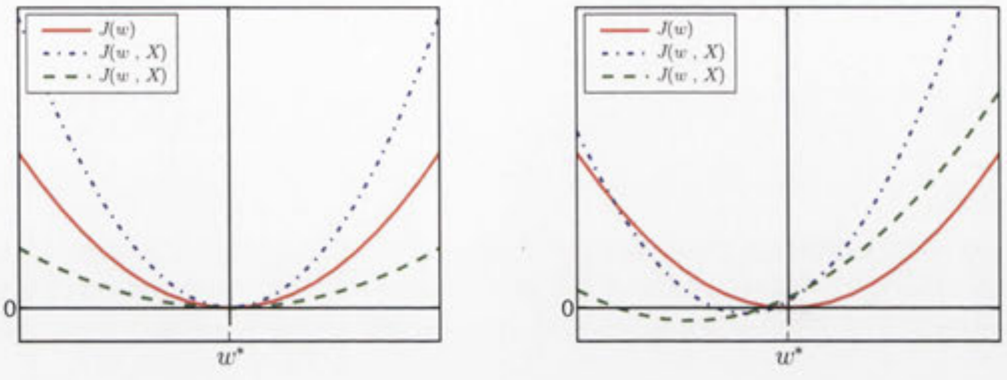


**Figure 5.1:** One-dimensional deterministic quadratic $J(w)$ (solid) and two stochastic approximations $J(w, X)$ (dash-dotted and dashed) obtained from different data samples $X$. Left: realizable; right: non-realizable.

## 5.5 Stochastic Quadratic Problems

We follow Schraudolph and Graepel (2003) in their choices of two stochastic quadratic (albeit ill-conditioned and semi-sparse) problems, which will be used in the experiments of Section 5.6 to illustrate the performance of various stochastic methods.

### 5.5.1 Deterministic Quadratic

The $d$-dimensional quadratic provides the simplest possible test setting that differentiates between various gradient methods. In its deterministic form, the objective function $J : \mathbb{R}^d \to \mathbb{R}$ is given by

$$J(\boldsymbol{w}) = \tfrac{1}{2}\,(\boldsymbol{w} - \boldsymbol{w}^*)^\top \boldsymbol{J}\boldsymbol{J}^\top\,(\boldsymbol{w} - \boldsymbol{w}^*)\,, \tag{5.19}$$

where $\boldsymbol{w}^* \in \mathbb{R}^d$ and $\boldsymbol{J} \in \mathbb{R}^{d \times d}$ are the optimal parameter vector and the Jacobian matrix, respectively, both of our choosing. Assuming that $\boldsymbol{J}$ has full rank, then by definition, the Hessian $\nabla^2 J(\boldsymbol{w}) = \boldsymbol{J}\boldsymbol{J}^\top$ is constant and positive definite here; the gradient is $\nabla J(\boldsymbol{w}) = \nabla^2 J(\boldsymbol{w})(\boldsymbol{w} - \boldsymbol{w}^*)$. Obviously, the minimal value of $J(\boldsymbol{w})$ is zero, *i.e.*, $J(\boldsymbol{w}^*) = 0$.

### 5.5.2 A Simple Stochastic Quadratic

A stochastic optimization problem analogous to the above can be defined by the data-dependent objective

$$J(\boldsymbol{w}, \boldsymbol{X}) = \frac{1}{2\,b}\,(\boldsymbol{w} - \boldsymbol{w}^*)^\top \boldsymbol{J}\boldsymbol{X}\boldsymbol{X}^\top \boldsymbol{J}^\top (\boldsymbol{w} - \boldsymbol{w}^*)\,, \tag{5.20}$$

where

$$\boldsymbol{X} := [\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots \boldsymbol{x}_b] \text{ with } \boldsymbol{x}_i \sim N(\boldsymbol{0}, \boldsymbol{I}),\ 1 \leq i \leq b \tag{5.21}$$

is a $d \times b$ matrix collecting a batch of $b$ random input vectors to the system, each drawn i.i.d. from a normal distribution. This means that $\mathbb{E}[\boldsymbol{X}\boldsymbol{X}^\top] = b\,\boldsymbol{I}$, hence in expectation the stochastic objective (5.20) is identical to the deterministic formulation (5.19):

$$\mathbb{E}_{\boldsymbol{X}}[J(\boldsymbol{w}, \boldsymbol{X})] = \frac{1}{2\,b}\,(\boldsymbol{w} - \boldsymbol{w}^*)^\top \boldsymbol{J}\ \mathbb{E}[\boldsymbol{X}\boldsymbol{X}^\top]\,\boldsymbol{J}^\top (\boldsymbol{w} - \boldsymbol{w}^*) = J(\boldsymbol{w}). \tag{5.22}$$

The optimization problem is harder here since the objective can only be probed by supplying stochastic inputs to the system, giving rise to the noisy estimates

$$\nabla_{\boldsymbol{w}} J(\boldsymbol{w}, \boldsymbol{X}) = \nabla^2_{\boldsymbol{w}} J(\boldsymbol{w}, \boldsymbol{X})(\boldsymbol{w} - \boldsymbol{w}^*) \text{ and } \nabla^2_{\boldsymbol{w}} J(\boldsymbol{w}, \boldsymbol{X}) = \frac{1}{b}\boldsymbol{J}\boldsymbol{X}\boldsymbol{X}^\top \boldsymbol{J}^\top \tag{5.23}$$

of the true gradient and Hessian, respectively. The degree of stochasticity is determined by the batch size $b$; the system becomes deterministic in the limit as $b \to \infty$.

Note that depending on the choice of $\boldsymbol{X}$, the matrix $(\boldsymbol{X}^\top \boldsymbol{J}^\top)$ can be rank deficient, meaning that the instantaneous Hessian $\nabla^2_{\boldsymbol{w}} J(\boldsymbol{w}, \boldsymbol{X})$ may not have full rank, and hence the solution to $\min_{\boldsymbol{w}} J(\boldsymbol{w}, \boldsymbol{X})$ may not be unique. Here we use $\operatorname{argmin}_{\boldsymbol{w}} J(\boldsymbol{w}, \boldsymbol{X})$ to denote the set of all possible solutions. For the stochastic quadratic problem (5.20), the solution set is given by

$$\operatorname*{argmin}_{\boldsymbol{w}} J(\boldsymbol{w}, \boldsymbol{X}) = \{\boldsymbol{w} : (\boldsymbol{w} - \boldsymbol{w}^*) \in \text{Null}(\boldsymbol{X}^\top \boldsymbol{J}^\top)\}. \tag{5.24}$$

where Null$(\cdot)$ denotes the null space of a matrix.

A one-dimensional view of (5.20) is given in Figure 5.1 (left) where we can see that $J(\boldsymbol{w}, \boldsymbol{X})$ is data-dependent but coincides with the deterministic objective function $J(\boldsymbol{w})$ at the optimum $\boldsymbol{w}^*$, *i.e.*, $\forall(\boldsymbol{X})\ J(\boldsymbol{w}^*, \boldsymbol{X}) = J(\boldsymbol{w}^*) = 0$. Furthermore, it follows from (5.24) that every stochastic approximation is minimized at $\boldsymbol{w}^*$:

$$\bigcap_{\boldsymbol{X}} \operatorname*{argmin}_{\boldsymbol{w}} J(\boldsymbol{w}, \boldsymbol{X}) \ = \ \{\boldsymbol{w}^*\}. \tag{5.25}$$

Schraudolph and Graepel (2003) call this *realizable*.

### 5.5.3 A Non-Realizable Quadratic

The stochastic quadratic (5.20) models realizable problems, *i.e.*, those where the solution set $\operatorname{argmin}_{\boldsymbol{w}} J(\boldsymbol{w}, \boldsymbol{X})$ contains the optimum $\boldsymbol{w}^*$ of the deterministic objective for all data samples. Of greater practical relevance are *non-realizable* problems in which $\boldsymbol{w}^*$ is not necessarily in the solution set, and the stochastic objective can be minimized at different points for different data samples, reflecting the conflicting demands placed on the model by the data. Following Schraudolph and Graepel (2003), we model this by incorporating, along with each data sample $\boldsymbol{X} \in \mathbb{R}^{d\times b}$ (5.21), an i.i.d. Gaussian random noise $\boldsymbol{\nu} \in \mathbb{R}^b$ with zero mean and variance $\mathbb{E}[\boldsymbol{\nu}\boldsymbol{\nu}^\top] = \sigma^2 \boldsymbol{I}$ into our objective:

$$\begin{aligned} &J(\boldsymbol{w}, \boldsymbol{X}, \boldsymbol{v}) = \frac{1}{2\,b}\, \mathbf{e}^\top \boldsymbol{e} - \frac{\sigma^2}{2}, \\ &\text{where } \;\boldsymbol{e} := \boldsymbol{X}^\top \boldsymbol{J}^\top (\boldsymbol{w} - \boldsymbol{w}^*) \;+\; \boldsymbol{\nu}\,. \end{aligned} \tag{5.26}$$

In expectation this is still identical to the deterministic quadratic (5.19):

$$\begin{aligned} \mathbb{E}_{\boldsymbol{X},\boldsymbol{\nu}}[J(\boldsymbol{w}, \boldsymbol{X}, \boldsymbol{v})] \;&=\; \mathbb{E}_{\boldsymbol{X}}\, J(\boldsymbol{w}, \boldsymbol{X}) + \frac{1}{b}(\boldsymbol{w} - \boldsymbol{w}^*)^\top \boldsymbol{J}\, \mathbb{E}_{\boldsymbol{X},\boldsymbol{\nu}}[\boldsymbol{X}\boldsymbol{v}] \\ &\quad + \frac{1}{2\,b}\,\mathbb{E}\left[\boldsymbol{\nu}^\top\boldsymbol{\nu}\right] - \frac{\sigma^2}{2} \\ &=\; J(\boldsymbol{w}) + 0 + \frac{\sigma^2}{2} - \frac{\sigma^2}{2} \\ &=\; J(\boldsymbol{w}), \end{aligned} \tag{5.27}$$

where $J(\boldsymbol{w}, \boldsymbol{X})$ is the realizable stochastic quadratic defined in (5.20). The presence of $\boldsymbol{\nu}$ makes it impossible to determine $\boldsymbol{w}^*$ precisely from a finite set of data samples; Figure 5.1 (right) shows that in this case $\operatorname{argmin}_{\boldsymbol{w}} J(\boldsymbol{w}, \boldsymbol{X}, \boldsymbol{v})$ does not necessarily contain $\boldsymbol{w}^*$. On the other hand, taking expectation of the stochastic gradient

$$\begin{aligned} \mathbb{E}_{\boldsymbol{X},\boldsymbol{\nu}}[\nabla_{\boldsymbol{w}} J(\boldsymbol{w}, \boldsymbol{X})] \;&=\; \frac{1}{b}\,\mathbb{E}_{\boldsymbol{X},\boldsymbol{\nu}}\left[\boldsymbol{J}\boldsymbol{X}\boldsymbol{X}^\top\boldsymbol{J}^\top(\boldsymbol{w} - \boldsymbol{w}^*) + \boldsymbol{J}\boldsymbol{X}\boldsymbol{\nu}\right] \\ &=\; \frac{1}{b}\left(\boldsymbol{J}\,\mathbb{E}\left[\boldsymbol{X}\boldsymbol{X}^\top\right]\boldsymbol{J}^\top(\boldsymbol{w} - \boldsymbol{w}^*) + \boldsymbol{J}\,\mathbb{E}_{\boldsymbol{X},\boldsymbol{\nu}}[\boldsymbol{X}\boldsymbol{\nu}]\right) \\ &=\; \boldsymbol{J}\boldsymbol{J}^\top(\boldsymbol{w} - \boldsymbol{w}^*) + \mathbf{0} \\ &=\; \nabla J(\boldsymbol{w}) \end{aligned} \tag{5.28}$$

reveals that it matches the true gradient. Therefore, we can use a step size decay schedule such as (5.4) to effectively average the noisy gradient estimates over progressively larger stretches of data, and hence in the limit obtain an estimate of the true gradient.

### 5.5.4 Choice of Jacobian

For our experiments, we choose the Jacobian $\boldsymbol{J}$ such that the Hessian has (1) eigenvalues of widely differing magnitude (ill-conditioning) and (2) eigenvectors with an intermediate degree of sparsity. We achieve this by imposing some sparsity on the notoriously ill-conditioned *Hilbert matrix*:

$$\boldsymbol{J}_{ij} := \begin{cases} \frac{1}{i+j-1} & \text{if } i \bmod j = 0 \\ & \text{or } j \bmod i = 0\,; \\ 0 & \text{otherwise}\,. \end{cases} \tag{5.29}$$

We use unconstrained online minimization of (5.20) and (5.26),[2] with $\boldsymbol{J}$ given by (5.29) in $d = 5$ dimensions, as our model problems for stochastic gradient methods. The condition number of the Hessian $\nabla^2 J(\boldsymbol{w})$ is $4.9 \times 10^3$.

## 5.6 Experiments

We now apply the past stochastic methods reviewed in Section 5.2 and our o(L)BFGS to the two stochastic quadratic problems just introduced. Their performance at different batch sizes is measured by the average number of data points needed to reach a pre-specified value of the deterministic objective (5.19). We also show the convergence behaviour of each algorithm at its optimal batch size by plotting the average deterministic objective value *vs.* the number of data points. As the performance of SMD essentially equals that of SGD in our experiments, it is not shown in the figures.

In all experiments we found it sufficient to set o(L)BFGS' free parameters $\epsilon$ and $\lambda$ to $10^{-10}$ and 0, respectively. $c$ was moderately tuned for good performance of oBFGS, and fixed to 0.1 throughout. We ran oLBFGS with the buffer sizes $m = 4$ and 10 (denoted oLBFGS *resp.* oLBFGS' in the figures). All experiments were run with varying batch size $b$, taking values in the set $\{4^i : i = 0, 1, \cdots, 8\}$.

### 5.6.1 Results on the Realizable Quadratic

Our first set of experiments were carried out on the realizable quadratic (5.20). For this simple problem, a constant step size proved sufficient. All methods used $\eta_t = b/(b+2)$ (tuned for good performance of SGD) except for NG, which required (5.4) with $\eta_0 = 1$ and $\tau = 100$.

Figure 5.2 shows that SGD suffers from slow convergence on this ill-conditioned problem: in comparison to other stochastic methods SGD (solid diamonds) needs to

[2] We shift down the non-realizable stochastic quadratic of Schraudolph and Graepel (2003) by $\sigma^2/2$ so as to establish (5.27).

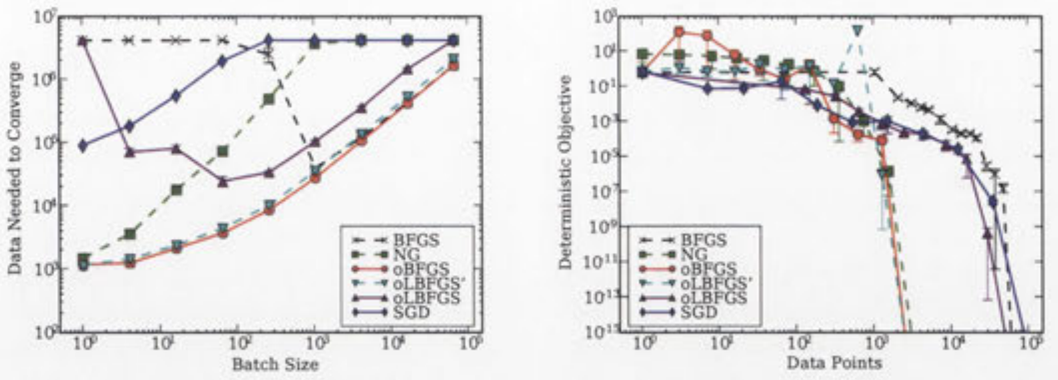


**Figure 5.2:** Average performance (with standard errors) of stochastic gradient methods on the realizable quadratic (5.20) over 10 matched random replications. Left: number of data points (up to a limit of $2^{22}$) needed to reach $J(\boldsymbol{w}) \leq 10^{-15}$ *vs.* batch size $b$. Right: deterministic loss $J(\boldsymbol{w})$ *vs.* number of data points seen at the optimal batch size for each algorithm.

see many more data points to decrease the deterministic objective value to a given threshold ($10^{-15}$). NG (dashed squares) benefits from its incorporation of second-order information, and hence greatly outperforms SGD here for $b < 10^3$. Note that the use of a running average in NG's estimate (5.10) of Riemannian metric tensor is optimal for this quadratic model, where the Hessian of $J(\boldsymbol{w})$ is constant.

We implemented a variant of oBFGS (denoted BFGS in the figures) that takes inconsistent gradient measurements (5.17). Figure 5.2 illustrates the disastrous consequences: this variant (dashed crosses) diverges for all batch sizes less than $10^3$ (Figure 5.2, left); at the optimal batch size, it is only marginally better than SGD (Figure 5.2, right). Note that Figure 5.2 (right) essentially shows the worst-case performance of each method because the average of deterministic objective values shown on log scale is dominated by the maximal value of them: for instance, in the worst case BFGS requires about $6 \cdot 10^4$ data points to reach $J(\boldsymbol{w}) \leq 10^{-15}$, hence the average deterministic objective shown in Figure 5.2 (right) is dominated by this worst run, while we can see in Figure 5.2 (left) that on average, BFGS requires around $4 \cdot 10^4$ data points to reach the given threshold.

While properly implemented, oBFGS (solid disks) outperforms NG for all batch sizes. OLBFGS with $m = 4$ (solid triangles) performs well down to $b \approx 100$ but degrades for smaller batch sizes. This is not surprising considering that the curvature estimate is now based on only 4 noisy measurements of the objective. Fortunately, the situation improves rapidly with increasing buffer size: for $m = 10$ the performance of oLBFGS (dashed triangles) is close to that of full online BFGS for all batch sizes.[3]

[3] Note that for $m > d$ LBFGS is computationally more expensive than full BFGS. For higher-dimensional problems, however, the beneficial effect of increasing $m$ will be realized well before approaching this anomalous regime.

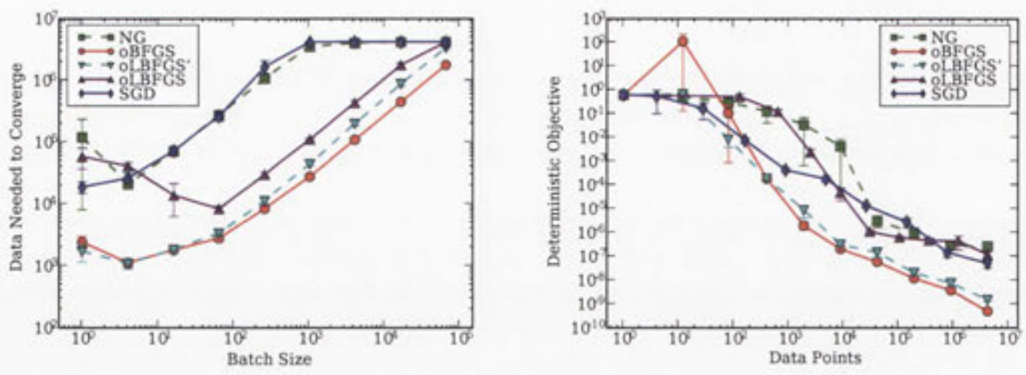


**Figure 5.3:** Average performance (with standard errors) of stochastic gradient methods on the non-realizable quadratic (5.26) over 10 matched random replications. Left: number of data points (up to a limit of $2^{22}$) needed to converge to $J(\boldsymbol{w}) < 10^{-5}$ *vs.* batch size $b$. Right: deterministic loss $J(\boldsymbol{w})$ *vs.* number of data points seen at the optimal batch size for each algorithm.

### 5.6.2 Results on the Non-Realizable Quadratic

We now turn to the more challenging non-realizable stochastic quadratic (5.26) with $\sigma = 10^{-2}$. In this set of experiments all methods used (5.4) with $\eta_0$ and $\tau$ moderately tuned for fast convergence at small batch sizes. The initial step size $\eta_0$ was set to $b/(b+2)$ for all methods except for NG and oLBFGS with $m = 4$ which required $\eta_0 = 0.04$ and $0.1 \cdot b/(b+2)$, respectively. $\tau$ was set to $10^4$ for SGD, 20 for NG and oBFGS, and $2{\cdot}10^4$ *resp.* 10 for oLBFGS with $m = 4$ *resp.* 10.

Figure 5.3 (left) shows the average number of data points needed to reach $J(\boldsymbol{w}) \leq 10^{-5}$, while the performance of each algorithm at its optimal batch size is illustrated in Figure 5.3 (right). Because the noise term $\boldsymbol{\nu}$ inflates the metric tensor (5.10), NG overestimates the Hessian, and ends up performing no better than SGD here. OBFGS, by contrast, bases its curvature estimate on *differences* of gradient measurements; as long as these are consistent (Section 5.3.3), any data-dependent noise or bias terms will thus be cancelled out. Consequently, oBFGS greatly outperforms both SGD and NG, converging about 20 times faster to $J(\boldsymbol{w}) = 10^{-5}$ at the convenient mini-batch size of $b = 4$ (Figure 5.3, right). The performance of oLBFGS with small buffer ($m = 4$) degrades for batch sizes below $b = 64$; a more generous buffer ($m = 10$), however, restores it to the level of full oBFGS.

## 5.7 Discussion

We developed stochastic variants of the BFGS and LBFGS quasi-Newton methods, suitable for online optimization of convex functions. Experiments on two stochastic quadratic problems show that our methods can greatly outperform past stochastic gradient algorithms, including a well-tuned natural gradient method. Unlike natural gradient, oLBFGS scales well to very high-dimensional problems, thanks to its matrix-free direction update.

Our online quasi-Newton methods require tuning of newly introduced free parameters ($\eta_t$, $c$, and $\lambda$ for oBFGS; $\eta_t$ and $\lambda$ for oLBFGS). Although no elaborate parameter tuning is needed, we expect further improvements from developing ways to automatically set and adapt these. One limitation of oLBFGS is that for very sparse data, oLBFGS may require a substantial buffer size $m$ to emulate a non-degenerate inverse curvature estimate.

Having established the utility of o(L)BFGS on the two synthetic stochastic quadratics, in the next chapter we will turn to more challenging and realistic convex optimization problems that stem from real-world applications.

Chapter 6

# Online LBFGS for the Training of Conditional Random Fields

In this chapter we apply our online LBFGS (oLBFGS) method (Section 5.4) to the training of Conditional Random Fields (CRFs) in natural language processing. We show that oLBFGS achieves state-of-the-art results on benchmark datasets in far fewer passes through the training data than its batch counterpart LBFGS. First we briefly review the problem formulation of CRF parameter estimation. The experimental results on three natural language processing datasets are reported in the subsequent sections.

## 6.1 Conditional Random Fields

Conditional Random Fields (CRFs) as a class of probabilistic models for labelling and parsing data have recently gained popularity in the machine learning community (Kumar and Hebert, 2004; Lafferty et al., 2001; Sha and Pereira, 2003). Parameter estimation in CRFs can be viewed as minimizing the negative log-posterior of the parameters $\boldsymbol{w} \in \mathbb{R}^d$ given the training data:

$$-\ln P(\boldsymbol{w}|\,\mathcal{X}, \mathcal{Z}), \tag{6.1}$$

where $\mathcal{X} := \{\boldsymbol{x}_i\}_{i=1}^n$ and $\mathcal{Z} := \{\boldsymbol{z}_i\}_{i=1}^n$ are the sets of feature vectors $\boldsymbol{x}_i$ and the corresponding label vectors $\boldsymbol{z}_i$, respectively. Bayes' rule suggests

$$P(\boldsymbol{w}|\,\mathcal{X}, \mathcal{Z}) \propto P(\boldsymbol{w})\, P(\mathcal{Z}\,|\,\mathcal{X}; \boldsymbol{w}). \tag{6.2}$$

If an i.i.d. conditional exponential family distribution over labels is assumed, *i.e.*,

$$P(\mathcal{Z}\,|\,\mathcal{X}; \boldsymbol{w}) := \exp\left(\sum_{i=1}^{n} \left(\phi(\boldsymbol{x}_i, \boldsymbol{z}_i)^\top \boldsymbol{w} - Z(\boldsymbol{w}, \boldsymbol{x}_i)\right)\right), \tag{6.3}$$

where $\phi(\cdot,\cdot)$ gives a vector of sufficient statistics that encodes features of the training data, and $Z(\cdot,\cdot)$ is the log-partition function

$$Z(\boldsymbol{w},\boldsymbol{x}) := \ln \sum_{\boldsymbol{z}} \exp(\phi(\boldsymbol{x},\boldsymbol{z})^\top \boldsymbol{w}), \tag{6.4}$$

then we can translate (6.1) into

$$-\ln[P(\boldsymbol{w})\,P(\mathcal{Z}\,|\,\mathcal{X};\boldsymbol{w})] = -\ln P(\boldsymbol{w}) - \sum_{i=1}^{n}\left(\phi(\boldsymbol{x}_i,\boldsymbol{z}_i)^\top \boldsymbol{w} - Z(\boldsymbol{w},\boldsymbol{x}_i)\right). \tag{6.5}$$

In practice, an isotropic Gaussian prior with variance $\sigma^2 \boldsymbol{I}$ is often imposed on $\boldsymbol{w}$, *i.e.*,

$$P(\boldsymbol{w}) \propto \exp(-\frac{1}{2\sigma^2}\|\boldsymbol{w}\|^2), \tag{6.6}$$

turning (6.5) into the familiar Maximum a posteriori estimation formulation:

$$J(\boldsymbol{w}) := \frac{\|\boldsymbol{w}\|^2}{2\sigma^2} - \sum_{i=1}^{n}\left(\phi(\boldsymbol{x}_i,\boldsymbol{z}_i)^\top \boldsymbol{w} - Z(\boldsymbol{w},\boldsymbol{x}_i)\right). \tag{6.7}$$

This objective is convex since the log-partition function (6.4) is convex in $\boldsymbol{w}$ (Wainwright and Jordan, 2003), and so are the other terms.

Conventional algorithms for batch CRF training, *i.e.*, minimizing (6.7), include generalized iterative scaling (GIS), conjugate gradient (CG), and limited-memory BFGS (LBFGS) (Sha and Pereira, 2003). As shown by Bottou (2009) and Vishwanathan et al. (2006), first-order stochastic gradient methods routinely outperform the conventional batch algorithms, *e.g.*, LBFGS, by carrying out the parameter update on stochastic approximation of $J(\boldsymbol{w})$, which can be formulated as

$$J_t(\boldsymbol{w}) := \frac{b}{n}\frac{\|\boldsymbol{w}\|^2}{2\sigma^2} - \sum_{i=1}^{b}\left(\phi(\boldsymbol{x}_{bt+i},\boldsymbol{z}_{bt+i})^\top \boldsymbol{w} - Z(\boldsymbol{w},\boldsymbol{x}_{bt+i})\right), \tag{6.8}$$

where $b$ is the size of a mini-batch of data sampled from the training set. Summing $J_t(\boldsymbol{w})$ over all batches of data recovers the deterministic objective: $\sum_{t=0}^{\frac{n}{b}-1} J_t(\boldsymbol{w}) = J(\boldsymbol{w})$.

## 6.2 Experimental Tasks

We replicated three experiments by Vishwanathan et al. (2006) who apply 1-D chain CRFs to problems in natural language processing, using their software — an enhanced version of Taku Kudo's `CRF++`[1] code — and following them in setting Gaussian prior (6.6) parameter $\sigma$ to 1. We used their CRF features for the experiment of Section 6.2.1.

[1]It is available from `http://crfpp.sourceforge.net`.

**Table 6.1**: Datasets used in our experiments and the batch size $b$ for stochastic methods.

| Dataset | Training Set Size | Test Set Size | # of Features | $b$ |
|---|---|---|---|---|
| CoNLL-2000 | 8936 | 2012 | 330731 | 8 |
| BioNLP/NLPBA-2004 | 18546 | 3856 | 455142 | 6 |
| BioCreAtIvE | 7500 | 5000 | 382543 | 6 |

For the other two experiments, we used binary orthographic features introduced by Settles (2004) as well as features based on neighbouring words; the features used by Vishwanathan et al. (2006) to model the correlations between the current and previous labels were not used here because `CRF++` actually can not properly process this type of features, causing it to produce misleadingly high generalization performance.

In our experiments we study the convergence behavior of various optimizers by plotting *deterministic* objective value on the training data *vs.* the number of passes through it. The generalization performance in terms of the F-score on the test data is reported, and benchmarked against the best F-score found in the literature that is achieved by using CRFs. The F-score is the harmonic mean of the precision and the recall measurements:

$$\begin{aligned} &\text{F-score} := 2\,(\text{precision} \times \text{recall})/(\text{precision} + \text{recall}), \quad \text{where} \\ &\text{precision} := \text{TP}/(\text{TP}+\text{FP}) \quad \text{and} \quad \text{recall} := \text{TP}/(\text{TP}+\text{FN}). \end{aligned} \tag{6.9}$$

"TP", "FP", and "FN" stand for the number of the true positives, false positives, and false negatives, respectively.

Table 6.1 summarizes the three datasets used in our experiments and the batch sizes used by stochastic methods. Since these CRFs have many (over $10^5$) parameters (*i.e.*, the number of features), neither full BFGS nor natural gradient (Section 5.2.3) can be used. However, we can apply our online LBFGS algorithm here: since the CRF parameter estimation problem is convex, the non-negativity condition (5.12) holds, and thus guarantees non-negativity of the inverse Hessian estimate emulated by oLBFGS. Our control methods are batch LBFGS (Algorithm 2.2) as supplied by `CRF++`[2] and SMD (Section 5.2.2), which is the best performing stochastic method in (Vishwanathan et al., 2006); their implementation of SGD (SGD as in Section 5.2.1 with fixed step sizes $\eta_t = 0.1$) is only shown for our first experiment since in the other cases it performed worse than SMD, while producing heavy oscillations that would have obscured our figures.

To cope with regions of low curvature, we employed a model-trust region parameter

[2]We note that the line search used by `CRF++` for LBFGS does not guarantee a monotonic decrease in the objective function value.

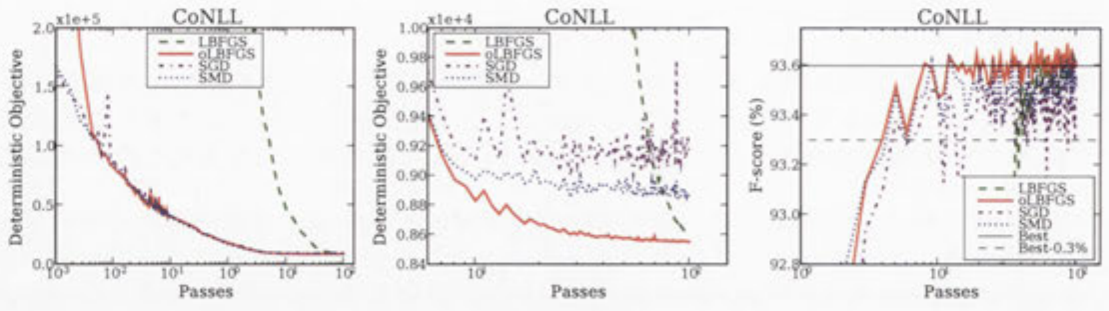


**Figure 6.1:** Performance of optimization algorithms plotted against number of passes through the CoNLL-2000 dataset. Left, center: deterministic objective on the training set; right: F-score on the test set.

$\lambda$ for oLBFGS (*cf.* Line 10 of Algorithm 5.1); we found it sufficient to set $\lambda = 1$ for our experiments. The step size of oLBFGS was determined by the decay schedule (5.4) with tuning parameters $\eta_0$ and $\tau$. We found empirically that setting $\eta_0 = 1$ and $\tau = 10^4$ typically leads to satisfactory results, and hence used this setting throughout. Free parameters of SMD were moderately tuned for good performance: $\boldsymbol{\eta}_0 = 0.1 \times \mathbf{1}$, $\mu = 0.02$ throughout, $\lambda = 0.99$ for the first experiment, and 0.1 for the others. The limited-memory buffer size $m = 5$ was used for both LBFGS and oLBFGS. To prevent stochastic methods from overfiting correlations across training instances, we randomly permuted the training data before each complete pass over it (*cf.* the two-step sampling procedure in Section 5.1). All methods were stopped after 100 full passes through the training data.

### 6.2.1 CoNLL-2000 Base NP Chunking

Our first experiment is the CoNLL-2000 Base NP chunking task (Sang and Buchholz, 2000). Text chunking as an intermediate step towards full parsing divides a text into syntactically correlated chunks of words. Each word in the training sentences is annotated automatically with part-of-speech (POS) tags. The task is to label each word with a label indicating whether it lies outside, starts, or continues a chunk.

Figure 6.1 (left) shows that oLBFGS (solid) initially is slower than SGD (dash dotted) and SMD (dotted). However, as shown in the zoomed-in figure (Figure 6.1, center), oLBFGS surpasses both SGD and SMD after 7 passes over the training data, and asymptotically achieves the lowest objective value of all methods. Moreover, Figure 6.1 (center) shows that after about 30 passes over the data, oLBFGS already reaches the final objective value obtained by the batch LBFGS method (dashed).

The best F-score on the test set (solid horizontal line in Figure 6.1, right) was reported to be 93.6% (Vishwanathan et al., 2006). Figure 6.1 (right) shows that LBFGS

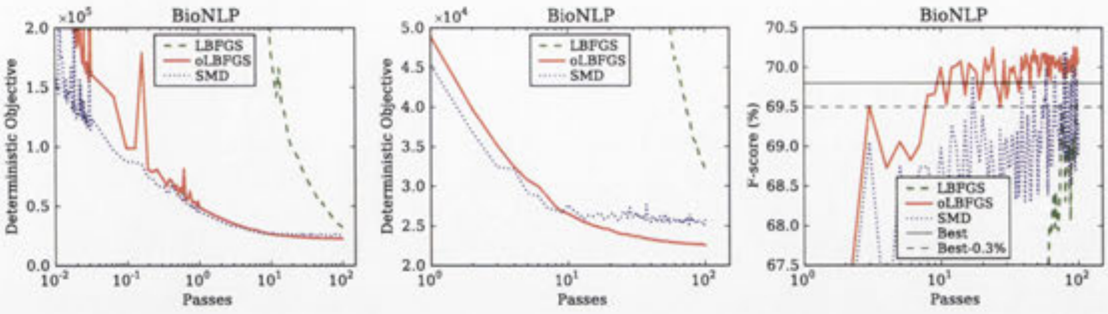


**Figure 6.2:** Performance of optimization algorithms plotted against number of passes through the BioNLP/NLPBA-2004 dataset. Left, center: deterministic objective on the training set; right: F-score on the test set.

requires 40 passes over the data to achieve a comparable generalization performance (F-score of 93.3%, dashed horizontal line), while SMD and SGD do so in 7 passes, oLBFGS in 6, though in several later passes SGD oscillates to a lower level.

### 6.2.2 BioNLP/NLPBA-2004 Shared Task

The BioNLP/NLPBA-2004 shared task (Kim et al., 2004) involves biomedical named-entity recognition on the GENIA corpus, aiming to identify and classify molecular biology terms in sentences of MEDLINE abstracts.

As can be seen in Figure 6.2 (left and center), both stochastic methods not only decrease the objective value substantially faster than the batch LBFGS method, but also asymptotically converge to a lower objective value: oLBFGS and SMD obtain the deterministic objective value eventually reached by LBFGS in more than one order of magnitude less passes over the data. In this task, oLBFGS again asymptotically outperforms SMD (Figure 6.2, center).

The best F-score reported by Settles (2004) was 69.8%.[3] oLBFGS reliably surpasses this F-score (Figure 6.2, right), while LBFGS and SMD do not: LBFGS eventually obtains an F-score of just over 69.5; the F-score of SMD settles around 68.5%-70%.

### 6.2.3 BioCreAtIvE Challenge Task 1A

The BioCreAtIvE challenge task 1A (Hirschman et al., 2005) is also a biomedical named-entity identification task. It focuses on gene and protein name identification in sentences of MEDLINE abstracts.

Similar to the previous set of experiments, both stochastic methods significantly outperform LBFGS in minimizing the objective; asymptotically, oLBFGS achieves the

[3]Settles (2005) later reported a better F-score of 70.5% for slightly improved features.

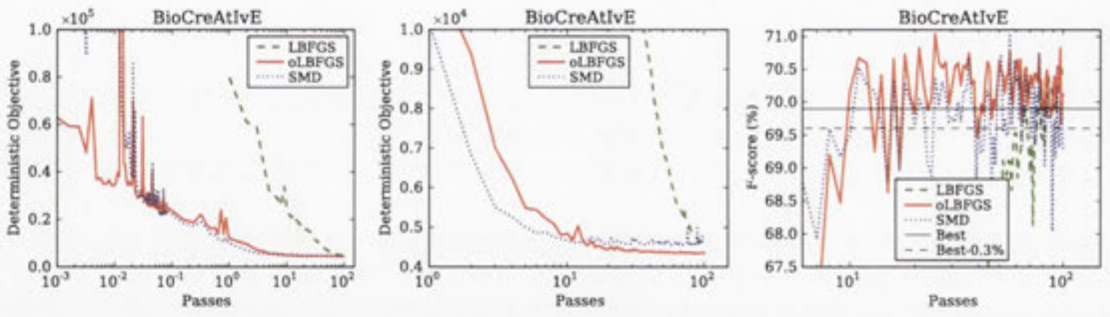


**Figure 6.3:** Performance of optimization algorithms plotted against number of passes through the BioCreAtIve dataset. Left, center: deterministic objective on the training set; right: F-score on the test set.

lowest objective value of all methods (Figure 6.3, left and center). On this dataset, oLBFGS reliable surpasses an F-score of 69.6% — 0.3% away from the F-score (69.9%) reported by Settles (2005) — after around 40 passes through the data, still faster than LBFGS (80 passes). Significant oscillations appear in the F-score of SMD, causing its poor performance here.

## 6.3 Discussion

We applied our online LBFGS algorithm to the training of CRFs in natural language processing. In all tasks, oLBFGS achieves state-of-the-art generalization results in far fewer passes through the data than LBFGS, while requiring only moderate tuning effort for its free parameters, with $\eta_0$, $\tau$, and $\lambda$ all fixed to their default values throughout. The generalization performance of oLBFGS is comparable to or better than existing stochastic methods, and the asymptotic objective value achieved by oLBFGS is consistently the lowest among stochastic methods.

Chapter 7

# Conclusions

This chapter concludes this thesis with a summary of key contributions and possible directions for future research.

## 7.1 Summary of Contributions

The key contributions of this thesis are:

1. A principled and robust BFGS quasi-Newton optimization method (subBFGS, Algorithm 3.1) and its limited-memory variant (subLBFGS) that are specifically designed for nonsmooth convex optimization problems in machine learning. SubBFGS is proven to globally converge to the optimal objective value under some technical conditions. SubLBFGS demonstrates competitive performance when benchmarked against specialized state-of-the-art machine learning solvers.

2. A subgradient reformulation (3.17 and 3.18) of the standard Wolfe conditions that can be used by an inexact line search to effectively reduce the value of a nonsmooth objective function in a given descent search direction.

3. An iterative direction-finding procedure (Algorithm 3.2) that is guaranteed to find a descent direction at a non-optimal position. This procedure can be plugged into any nonsmooth optimization algorithm which requires a descent direction for its parameter update.

4. A new efficient algorithm (Algorithm 3.4) for identifying the nonsmooth points of a one-dimensional pointwise maximum of linear functions.

5. Exact line search methods specialized for $L_2$-regularized risk minimization with the binary hinge loss (Algorithm 3.3) and its generalizations to the multiclass and multilabel settings (Algorithm 3.5). These methods can be used as black-box procedures to accelerate the convergence of any adaptive classifier whose parameter update takes the form of (1.6).

6. Stochastic variants of standard BFGS (Algorithm 5.1), in both its full and memory-limited forms, for online optimization of convex functions. Both algorithms are scalable to large datasets. Online LBFGS is, in addition, also scalable to large models with many parameters.

## 7.2 Future Research Directions

In the following we suggest several possible extensions of the work presented in this thesis:

1. In many of our experiments of Chapter 4 we observe that subLBFGS decreases the objective function rapidly early on but slows down closer to the optimum. Bundle methods, by contrast, exhibit slow initial progress but tend to be competitive asymptotically. One promising research direction would be to develop hybrid optimizers that are able to combine the strength of these two methods, *e.g.*, by switching from sub(L)BFGS to a bundle method as appropriate.

2. This thesis demonstrates the use of sub(L)BFGS on nonsmooth regularized risk minimization problems where the nonsmoothness stems from piecewise linear terms in the objective function. Extending sub(L)BFGS to problems with other forms of nonsmoothness would be an interesting research topic. For instance, the objective functions of computational problems in multiview geometry (Hartley and Kahl, 2007; Kahl and Hartley, 2008) are the pointwise maximum of quartic ($4^{\text{th}}$-order) polynomials. These problems are nonsmooth at those intersections of polynomials where the maximum is attained. Moreover, they are not convex but quasi-convex, *i.e.*, they have convex sublevel sets. Adapting sub(L)BFGS to these circumstances could improve the dominant second-order cone programming (SOCP) approach to these problems.

3. The current version of online BFGS (*resp.* online LBFGS) is applicable only to convex problems, where it can maintain positivity of its curvature estimate $\boldsymbol{B}_t$. Extending o(L)BFGS to local optimization of non-convex objectives would be an interesting research topic to pursue.

4. Our stochastic variants of the BFGS and LBFGS methods require tuning of a few free parameters; in our experience, such tuning becomes critical for non-convex optimization problems. It would be very useful if principled ways could be developed to automatically set and adapt them.

Appendix A

# Appendix

## A.1 Bundle Search for a Descent Direction

Recall from Section 3.2.2 that at a subdifferential point $\boldsymbol{w}$ our goal is to find a descent direction $\boldsymbol{p}^*$ which minimizes the pseudo-quadratic model:[1]

$$M(\boldsymbol{p}) := \tfrac{1}{2}\boldsymbol{p}^\top \boldsymbol{B}^{-1}\boldsymbol{p} + \sup_{\boldsymbol{g}\in\partial J(\boldsymbol{w})} \boldsymbol{g}^\top \boldsymbol{p}. \tag{A.1}$$

This is generally intractable due to the presence of a supremum over the entire subdifferential $\partial J(\boldsymbol{w})$. We therefore propose a bundle-based descent direction finding procedure (Algorithm 3.2) which progressively approaches $M(\boldsymbol{p})$ from below via a series of convex functions $M^{(1)}(\boldsymbol{p}), \cdots, M^{(i)}(\boldsymbol{p})$, each taking the same form as $M(\boldsymbol{p})$ but with the supremum defined over a countable subset of $\partial J(\boldsymbol{w})$. At iteration $i$ our convex lower bound $M^{(i)}(\boldsymbol{p})$ takes the form

$$\begin{aligned} M^{(i)}(\boldsymbol{p}) &:= \tfrac{1}{2}\boldsymbol{p}^\top \boldsymbol{B}^{-1}\boldsymbol{p} + \sup_{\boldsymbol{g}\in\mathcal{V}^{(i)}} \boldsymbol{g}^\top \boldsymbol{p}, \text{ where} \\ \mathcal{V}^{(i)} &:= \{\boldsymbol{g}^{(j)} : j \le i,\ i,j\ \in \mathbb{N}\} \subseteq \partial J(\boldsymbol{w}). \end{aligned} \tag{A.2}$$

Given an iterate $\boldsymbol{p}^{(j-1)} \in \mathbb{R}^d$ we find a *violating subgradient* $\boldsymbol{g}^{(j)}$ via

$$\boldsymbol{g}^{(j)} := \operatorname*{arg\,sup}_{\boldsymbol{g}\in\partial J(\boldsymbol{w})} \boldsymbol{g}^\top \boldsymbol{p}^{(j-1)}. \tag{A.3}$$

Violating subgradients recover the true objective $M(\boldsymbol{p})$ at the iterates $\boldsymbol{p}^{(j-1)}$:

$$M(\boldsymbol{p}^{(j-1)}) = M^{(j)}(\boldsymbol{p}^{(j-1)}) = \tfrac{1}{2}\boldsymbol{p}^{(j-1)\top}\boldsymbol{B}^{-1}\boldsymbol{p}^{(j-1)} + \boldsymbol{g}^{(j)\top}\boldsymbol{p}^{(j-1)}. \tag{A.4}$$

To produce the iterates $\boldsymbol{p}^{(i)}$, we rewrite $\min_{\boldsymbol{p}\in\mathbb{R}^d} M^{(i)}(\boldsymbol{p})$ as a constrained optimiza-

[1] For ease of exposition we are suppressing the iteration index $t$ here.

tion problem (3.13), which allows us to write the Lagrangian of (A.2) as

$$L^{(i)}(\boldsymbol{p}, \xi, \boldsymbol{\alpha}) := \tfrac{1}{2}\boldsymbol{p}^\top \boldsymbol{B}^{-1}\boldsymbol{p} + \xi - \boldsymbol{\alpha}^\top(\xi\mathbf{1} - {\boldsymbol{G}^{(i)}}^\top \boldsymbol{p}), \tag{A.5}$$

where $\boldsymbol{G}^{(i)} := [\boldsymbol{g}^{(1)}, \boldsymbol{g}^{(2)}, \ldots, \boldsymbol{g}^{(i)}] \in \mathbb{R}^{d\times i}$ collects past violating subgradients, and $\boldsymbol{\alpha}$ is a column vector of non-negative Lagrange multipliers. Setting the derivative of (A.5) with respect to the primal variables $\xi$ and $\boldsymbol{p}$ to zero yields, respectively,

$$\boldsymbol{\alpha}^\top \mathbf{1} = 1 \quad \text{and} \tag{A.6}$$

$$\boldsymbol{p} = -\boldsymbol{B}\boldsymbol{G}^{(i)}\boldsymbol{\alpha}. \tag{A.7}$$

The primal variable $\boldsymbol{p}$ and the dual variable $\boldsymbol{\alpha}$ are related via the dual connection (A.7). To eliminate the primal variables $\xi$ and $\boldsymbol{p}$, we plug (A.6) and (A.7) back into the Lagrangian to obtain the dual of $M^{(i)}(\boldsymbol{p})$:

$$\begin{aligned} D^{(i)}(\boldsymbol{\alpha}) &:= -\tfrac{1}{2}(\boldsymbol{G}^{(i)}\boldsymbol{\alpha})^\top \boldsymbol{B}(\boldsymbol{G}^{(i)}\boldsymbol{\alpha}) \\ &\text{s.t. } \boldsymbol{\alpha} \in [0,1]^i, \ \|\boldsymbol{\alpha}\|_1 = 1. \end{aligned} \tag{A.8}$$

The dual objective $D^{(i)}(\boldsymbol{\alpha})$ (*resp.* primal objective $M^{(i)}(\boldsymbol{p})$) can be maximized (*resp.* minimized) exactly via quadratic programming. However, doing so may incur substantial computational expense. Instead we adopt an iterative scheme which is cheap and easy to implement yet guarantees dual improvement.

Let $\boldsymbol{\alpha}^{(i)} \in [0,1]^i$ be a feasible solution for $D^{(i)}(\boldsymbol{\alpha})$.[2] The corresponding primal solution $\boldsymbol{p}^{(i)}$ can be found by using (A.7). This in turn allows us to compute the next violating subgradient $\boldsymbol{g}^{(i+1)}$ via (A.3). With the new violating subgradient the dual becomes

$$\begin{aligned} D^{(i+1)}(\boldsymbol{\alpha}) \ &:= -\tfrac{1}{2}(\boldsymbol{G}^{(i+1)}\boldsymbol{\alpha})^\top \boldsymbol{B}(\boldsymbol{G}^{(i+1)}\boldsymbol{\alpha}) \\ &\text{s.t. } \ \boldsymbol{\alpha} \in [0,1]^{i+1}, \ \|\boldsymbol{\alpha}\|_1 = 1, \end{aligned} \tag{A.9}$$

where the subgradient matrix is now extended:

$$\boldsymbol{G}^{(i+1)} = [\boldsymbol{G}^{(i)}, \boldsymbol{g}^{(i+1)}]. \tag{A.10}$$

Our iterative strategy constructs a new feasible solution $\boldsymbol{\alpha} \in [0,1]^{i+1}$ for (A.9) by

[2]Note that $\boldsymbol{\alpha}^{(1)} = \mathbf{1}$ is a feasible solution for $D^{(1)}(\boldsymbol{\alpha})$.

constraining it to take the following form:

$$\boldsymbol{\alpha} = \begin{bmatrix} (1-\mu)\boldsymbol{\alpha}^{(i)} \\ \mu \end{bmatrix}, \quad \text{where} \quad \mu \in [0,1]. \tag{A.11}$$

In other words, we maximize a one-dimensional function $\bar{D}^{(i+1)} : [0,1] \to \mathbb{R}$:

$$\begin{aligned} \bar{D}^{(i+1)}(\mu) &:= -\tfrac{1}{2}\left(\boldsymbol{G}^{(i+1)}\boldsymbol{\alpha}\right)^\top \boldsymbol{B}\left(\boldsymbol{G}^{(i+1)}\boldsymbol{\alpha}\right) \qquad (A.12)\\ &= -\tfrac{1}{2}\left((1-\mu)\bar{\boldsymbol{g}}^{(i)} + \mu\boldsymbol{g}^{(i+1)}\right)^\top \boldsymbol{B}\left((1-\mu)\bar{\boldsymbol{g}}^{(i)} + \mu\boldsymbol{g}^{(i+1)}\right), \end{aligned}$$

where

$$\bar{\boldsymbol{g}}^{(i)} := \boldsymbol{G}^{(i)}\boldsymbol{\alpha}^{(i)} \in \partial J(\boldsymbol{w}) \tag{A.13}$$

lies in the convex hull of $\boldsymbol{g}^{(j)} \in \partial J(\boldsymbol{w})\ \forall j \le i$ (and hence in the convex set $\partial J(\boldsymbol{w})$) because $\boldsymbol{\alpha}^{(i)} \in [0,1]^i$ and $\|\boldsymbol{\alpha}^{(i)}\|_1 = 1$. Moreover, $\mu \in [0,1]$ ensures the feasibility of the dual solution. Noting that $\bar{D}^{(i+1)}(\mu)$ is a concave quadratic function, we set

$$\partial\bar{D}^{(i+1)}(\mu) = \left(\bar{\boldsymbol{g}}^{(i)} - \boldsymbol{g}^{(i+1)}\right)^\top \boldsymbol{B}\left((1-\eta)\bar{\boldsymbol{g}}^{(i)} + \eta\boldsymbol{g}^{(i+1)}\right) = 0 \tag{A.14}$$

to obtain the optimum

$$\mu^* := \operatorname*{argmax}_{\mu\in[0,1]} \bar{D}^{(i+1)}(\mu) = \min\left(1, \max\left(0, \frac{(\bar{\boldsymbol{g}}^{(i)} - \boldsymbol{g}^{(i+1)})^\top \boldsymbol{B}\bar{\boldsymbol{g}}^{(i)}}{(\bar{\boldsymbol{g}}^{(i)} - \boldsymbol{g}^{(i+1)})^\top \boldsymbol{B}(\bar{\boldsymbol{g}}^{(i)} - \boldsymbol{g}^{(i+1)})}\right)\right). \tag{A.15}$$

Our dual solution at step $i+1$ then becomes

$$\boldsymbol{\alpha}^{(i+1)} := \begin{bmatrix} (1-\mu^*)\boldsymbol{\alpha}^{(i)} \\ \mu^* \end{bmatrix}. \tag{A.16}$$

Furthermore, from (A.10), (A.11), and (A.13) it follows that $\bar{\boldsymbol{g}}^{(i)}$ can be maintained via an incremental update (Line 8 of Algorithm 3.2):

$$\bar{\boldsymbol{g}}^{(i+1)} := \boldsymbol{G}^{(i+1)}\boldsymbol{\alpha}^{(i+1)} = (1-\mu^*)\bar{\boldsymbol{g}}^{(i)} + \mu^*\boldsymbol{g}^{(i+1)}, \tag{A.17}$$

which combined with the dual connection (A.7) yields an incremental update for the primal solution (Line 9 of Algorithm 3.2):

$$\begin{aligned} \boldsymbol{p}^{(i+1)} := -\boldsymbol{B}\bar{\boldsymbol{g}}^{(i+1)} &= -(1-\mu^*)\boldsymbol{B}\bar{\boldsymbol{g}}^{(i)} - \mu^*\boldsymbol{B}\boldsymbol{g}^{(i+1)} \\ &= (1-\mu^*)\boldsymbol{p}^{(i)} - \mu^*\boldsymbol{B}\boldsymbol{g}^{(i+1)}. \end{aligned} \tag{A.18}$$

Using (A.17) and (A.18), computing a primal solution (Lines 7–9 of Algorithm 3.2) costs a total of $O(d^2)$ time (*resp.* $O(md)$ time for LBFGS with buffer size $m$), where $d$ is the dimensionality of the optimization problem. Note that maximizing $D^{(i+1)}(\boldsymbol{\alpha})$ directly via quadratic programming generally results in a larger progress than that obtained by our approach.

In order to measure the quality of our solution at iteration $i$, we define the quantity

$$\epsilon^{(i)} := \min_{j\leq i} M^{(j+1)}(\boldsymbol{p}^{(j)}) - D^{(i)}(\boldsymbol{\alpha}^{(i)}) = \min_{j\leq i} M(\boldsymbol{p}^{(j)}) - D^{(i)}(\boldsymbol{\alpha}^{(i)}), \tag{A.19}$$

where the second equality follows directly from (A.4). Let $D(\boldsymbol{\alpha})$ be the corresponding dual problem of $M(\boldsymbol{p})$, with the property $D\left(\begin{bmatrix}\boldsymbol{\alpha}^{(i)}\\ \mathbf{0}\end{bmatrix}\right) = D^{(i)}(\boldsymbol{\alpha}^{(i)})$, and let $\boldsymbol{\alpha}^*$ be the optimal solution to $\operatorname{argmax}_{\boldsymbol{\alpha}\in\mathcal{A}} D(\boldsymbol{\alpha})$ in some domain $\mathcal{A}$ of interest. As a consequence of the weak duality theorem (Hiriart-Urruty and Lemaréchal, 1993, Theorem XII.2.1.5), $\min_{\boldsymbol{p}\in\mathbb{R}^d} M(\boldsymbol{p}) \geq D(\boldsymbol{\alpha}^*)$. Therefore (A.19) implies that

$$\epsilon^{(i)} \geq \min_{\boldsymbol{p}\in\mathbb{R}^d} M(\boldsymbol{p}) - D^{(i)}(\boldsymbol{\alpha}^{(i)}) \geq \min_{\boldsymbol{p}\in\mathbb{R}^d} M(\boldsymbol{p}) - D(\boldsymbol{\alpha}^*) \geq 0. \tag{A.20}$$

The second inequality essentially says that $\epsilon^{(i)}$ is an upper bound on the duality gap. In fact, Theorem A.2.3 below shows that $(\epsilon^{(i)} - \epsilon^{(i+1)})$ is bounded away from 0, *i.e.*, $\epsilon^{(i)}$ is monotonically decreasing. This guides us to design a practical stopping criterion (Line 6 of Algorithm 3.2) for our direction-finding procedure. Furthermore, using the dual connection (A.7), we can derive an implementable formula for $\epsilon^{(i)}$:

$$\begin{aligned}
\epsilon^{(i)} &= \min_{j\leq i}\left[\tfrac{1}{2}\boldsymbol{p}^{(j)\top}\boldsymbol{B}^{-1}\boldsymbol{p}^{(j)} + \boldsymbol{p}^{(j)\top}\boldsymbol{g}^{(j+1)} + \tfrac{1}{2}(\boldsymbol{G}^{(i)}\boldsymbol{\alpha}^{(i)})^\top\boldsymbol{B}(\boldsymbol{G}^{(i)}\boldsymbol{\alpha}^{(i)})\right]\\
&= \min_{j\leq i}\left[-\tfrac{1}{2}\boldsymbol{p}^{(j)\top}\bar{\boldsymbol{g}}^{(j)} + \boldsymbol{p}^{(j)\top}\boldsymbol{g}^{(j+1)} - \tfrac{1}{2}\boldsymbol{p}^{(i)\top}\bar{\boldsymbol{g}}^{(i)}\right]\\
&= \min_{j\leq i}\left[\boldsymbol{p}^{(j)\top}\boldsymbol{g}^{(j+1)} - \tfrac{1}{2}(\boldsymbol{p}^{(j)\top}\bar{\boldsymbol{g}}^{(j)} + {\boldsymbol{p}^{(i)}}^\top\bar{\boldsymbol{g}}^{(i)})\right],\\
&\text{where } \boldsymbol{g}^{(j+1)} := \operatorname*{arg\,sup}_{\boldsymbol{g}\in\partial J(\boldsymbol{w})} \boldsymbol{g}^\top\boldsymbol{p}^{(j)} \text{ and } \bar{\boldsymbol{g}}^{(j)} := \boldsymbol{G}^{(j)}\boldsymbol{\alpha}^{(j)} \ \forall j\leq i.
\end{aligned} \tag{A.21}$$

It is worth noting that continuous progress in the dual objective value does not necessarily prevent an increase in the primal objective value, *i.e.*, it is possible that $M(\boldsymbol{p}^{(i+1)}) \geq M(\boldsymbol{p}^{(i)})$. Therefore, we choose the best primal solution so far,

$$\boldsymbol{p} := \operatorname*{argmin}_{j\leq i} M(\boldsymbol{p}^{(j)}), \tag{A.22}$$

as the search direction (Line 18 of Algorithm 3.2) for the parameter update (2.5). This direction is a direction of descent as long as the last iterate $\boldsymbol{p}^{(i)}$ fulfills the descent condition (3.10). To see this, we use (A.32–A.34) below to get $\sup_{\boldsymbol{g}\in\partial J(\boldsymbol{w})} \boldsymbol{g}^\top\boldsymbol{p}^{(i)} =$

$M(\boldsymbol{p}^{(i)}) + D^{(i)}(\boldsymbol{\alpha}^{(i)})$, and since

$$M(\boldsymbol{p}^{(i)}) \geq \min_{j \leq i} M(\boldsymbol{p}^{(j)}) \text{ and } D^{(i)}(\boldsymbol{\alpha}^{(i)}) \geq D^{(j)}(\boldsymbol{\alpha}^{(j)}) \ \ \forall j \leq i, \tag{A.23}$$

definition (A.22) immediately gives $\sup_{\boldsymbol{g} \in \partial J(\boldsymbol{w})} \boldsymbol{g}^\top \boldsymbol{p}^{(i)} \geq \sup_{\boldsymbol{g} \in \partial J(\boldsymbol{w})} \boldsymbol{g}^\top \boldsymbol{p}$. Hence if $\boldsymbol{p}^{(i)}$ is a descent direction, then so is $\boldsymbol{p}$.

We now show that if the current parameter vector $\boldsymbol{w}$ is not optimal, then a direction-finding tolerance $\epsilon \geq 0$ exists for Algorithm 3.2 such that the returned search direction $\boldsymbol{p}$ is a descent direction, *i.e.*, $\sup_{\boldsymbol{g} \in \partial J(\boldsymbol{w})} \boldsymbol{g}^\top \boldsymbol{p} < 0$.

**Lemma A.1.1** *Let $\boldsymbol{B}$ be the current approximation to the inverse Hessian maintained by Algorithm 3.1, and $h > 0$ a lower bound on the eigenvalues of $\boldsymbol{B}$. If the current iterate $\boldsymbol{w}$ is not optimal: $\boldsymbol{0} \notin \partial J(\boldsymbol{w})$, and the number of direction-finding iterations is unlimited ($k_{max} = \infty$), then there exists a direction-finding tolerance $\epsilon \geq 0$ such that the descent direction $\boldsymbol{p} = -\boldsymbol{B}\bar{\boldsymbol{g}}$, $\bar{\boldsymbol{g}} \in \partial J(\boldsymbol{w})$ returned by Algorithm 3.2 at $\boldsymbol{w}$ satisfies $\sup_{\boldsymbol{g} \in \partial J(\boldsymbol{w})} \boldsymbol{g}^\top \boldsymbol{p} < 0$.*

**Proof** Algorithm 3.2 returns $\boldsymbol{p}$ after $i$ iterations when $\epsilon^{(i)} \leq \epsilon$, where $\epsilon^{(i)} = M(\boldsymbol{p}) - D^{(i)}(\boldsymbol{\alpha}^{(i)})$ by definitions (A.19) and (A.22). Using definition (A.8) of $D^{(i)}(\boldsymbol{\alpha}^{(i)})$, we have

$$-D^{(i)}(\boldsymbol{\alpha}^{(i)}) \ = \ \tfrac{1}{2}(\boldsymbol{G}^{(i)}\boldsymbol{\alpha}^{(i)})^\top \boldsymbol{B}(\boldsymbol{G}^{(i)}\boldsymbol{\alpha}^{(i)}) \ = \ \tfrac{1}{2}\bar{\boldsymbol{g}}^{(i)\top}\boldsymbol{B}\bar{\boldsymbol{g}}^{(i)}, \tag{A.24}$$

where $\bar{\boldsymbol{g}}^{(i)} = \boldsymbol{G}^{(i)}\boldsymbol{\alpha}^{(i)}$ is a subgradient in $\partial J(\boldsymbol{w})$. On the other hand, using (A.1) and (A.18), one can write

$$\begin{aligned} M(\boldsymbol{p}) \ &= \ \sup_{\boldsymbol{g} \in \partial J(\boldsymbol{w})} \boldsymbol{g}^\top \boldsymbol{p} \ + \ \tfrac{1}{2}\boldsymbol{p}^\top \boldsymbol{B}^{-1}\boldsymbol{p} \\ &= \ \sup_{\boldsymbol{g} \in \partial J(\boldsymbol{w})} \boldsymbol{g}^\top \boldsymbol{p} \ + \ \tfrac{1}{2}\bar{\boldsymbol{g}}^\top \boldsymbol{B}\bar{\boldsymbol{g}}, \quad \text{where} \quad \bar{\boldsymbol{g}} \in \partial J(\boldsymbol{w}). \end{aligned} \tag{A.25}$$

Putting together (A.24) and (A.25), and using $\boldsymbol{B} \succ h$, one obtains

$$\epsilon^{(i)} = \sup_{\boldsymbol{g} \in \partial J(\boldsymbol{w})} \boldsymbol{g}^\top \boldsymbol{p} + \tfrac{1}{2}\bar{\boldsymbol{g}}^\top \boldsymbol{B}\bar{\boldsymbol{g}} + \tfrac{1}{2}\bar{\boldsymbol{g}}^{(i)\top}\boldsymbol{B}\bar{\boldsymbol{g}}^{(i)} \geq \sup_{\boldsymbol{g} \in \partial J(\boldsymbol{w})} \boldsymbol{g}^\top \boldsymbol{p} + \frac{h}{2}\|\bar{\boldsymbol{g}}\|^2 + \frac{h}{2}\|\bar{\boldsymbol{g}}^{(i)}\|^2. \tag{A.26}$$

Since $\boldsymbol{0} \notin \partial J(\boldsymbol{w})$, the last two terms of (A.26) are strictly positive; and by (A.20), $\epsilon^{(i)} \geq 0$ . The claim follows by choosing an $\epsilon$ such that $(\forall i) \ \frac{h}{2}(\|\bar{\boldsymbol{g}}\|^2 + \|\bar{\boldsymbol{g}}^{(i)}\|^2) > \epsilon \geq \epsilon^{(i)} \geq 0$. ■

Using the notation from Lemma A.1.1, we show in the following corollary that a stricter upper bound on $\epsilon$ allows us to bound $\sup_{\boldsymbol{g} \in \partial J(\boldsymbol{w})} \boldsymbol{g}^\top \boldsymbol{p}$ in terms of $\bar{\boldsymbol{g}}^\top \boldsymbol{B}\bar{\boldsymbol{g}}$ and

$\|\bar{\boldsymbol{g}}\|$. This will be used in Appendix A.4 to establish the global convergence of the subBFGS algorithm.

**Corollary A.1.2** *Under the conditions of Lemma A.1.1, there exists an $\epsilon \geq 0$ for Algorithm 3.2 such that the search direction $\mathbf{p}$ generated by Algorithm 3.2 satisfies*

$$\sup_{\boldsymbol{g} \in \partial J(\boldsymbol{w})} \boldsymbol{g}^\top \boldsymbol{p} \leq -\tfrac{1}{2} \bar{\boldsymbol{g}}^\top \boldsymbol{B} \bar{\boldsymbol{g}} \leq -\frac{h}{2} \|\bar{\boldsymbol{g}}\|^2 < 0. \tag{A.27}$$

**Proof** Using (A.26), we have

$$(\forall i) \;\; \epsilon^{(i)} \geq \sup_{\boldsymbol{g} \in \partial J(\boldsymbol{w})} \boldsymbol{g}^\top \boldsymbol{p} + \tfrac{1}{2} \bar{\boldsymbol{g}}^\top \boldsymbol{B} \bar{\boldsymbol{g}} + \frac{h}{2} \|\bar{\boldsymbol{g}}^{(i)}\|^2. \tag{A.28}$$

The first inequality in (A.27) results from choosing an $\epsilon$ such that

$$(\forall i) \;\; \frac{h}{2} \|\bar{\boldsymbol{g}}^{(i)}\|^2 \geq \epsilon \geq \epsilon^{(i)} \geq 0. \tag{A.29}$$

The lower bound $h > 0$ on the spectrum of $\boldsymbol{B}$ yields the second inequality in (A.27), and the third follows from the fact that $\|\bar{\boldsymbol{g}}\| > 0$ at non-optimal iterates. ∎

## A.2 Convergence of the Descent Direction Search

Using the notation established in Appendix A.1, we now prove the convergence of Algorithm 3.2 via several technical intermediate steps. The proof shares similarities with the proofs found in Smola et al. (2007), Shalev-Shwartz and Singer (2008), and Warmuth et al. (2008). The key idea is that at each iterate Algorithm 3.2 decreases the upper bound $\epsilon^{(i)}$ on the distance from the optimality, and the decrease in $\epsilon^{(i)}$ is characterized by the recurrence $\epsilon^{(i)} - \epsilon^{(i+1)} \geq c(\epsilon^{(i)})^2$ with $c > 0$ (Theorem A.2.3). Analysing this recurrence then gives the convergence rate of the algorithm (Theorem A.2.5).

We first provide two technical lemmas (Lemma A.2.1 and A.2.2) that are needed to prove Theorem A.2.3.

**Lemma A.2.1** *Let $\bar{D}^{(i+1)}(\mu)$ be the one-dimensional function defined in (A.12), and $\epsilon^{(i)}$ the positive measure defined in (A.19). Then $\epsilon^{(i)} \leq \partial \bar{D}^{(i+1)}(0)$.*

**Proof** Let $\boldsymbol{p}^{(i)}$ be our primal solution at iteration $i$, derived from the dual solution $\boldsymbol{\alpha}^{(i)}$ using the dual connection (A.7). We then have

$$\boldsymbol{p}^{(i)} = -\boldsymbol{B} \bar{\boldsymbol{g}}^{(i)}, \quad \text{where} \quad \bar{\boldsymbol{g}}^{(i)} := \boldsymbol{G}^{(i)} \boldsymbol{\alpha}^{(i)}. \tag{A.30}$$

Definition (A.1) of $M(\boldsymbol{p})$ implies that

$$M(\boldsymbol{p}^{(i)}) = \tfrac{1}{2}\boldsymbol{p}^{(i)\top}\boldsymbol{B}^{-1}\boldsymbol{p}^{(i)} + \boldsymbol{p}^{(i)\top}\boldsymbol{g}^{(i+1)}, \tag{A.31}$$

where

$$\boldsymbol{g}^{(i+1)} := \operatorname*{arg\,sup}_{\boldsymbol{g}\in\partial J(\boldsymbol{w})} \boldsymbol{g}^\top \boldsymbol{p}^{(i)}. \tag{A.32}$$

Using (A.30), we have $\boldsymbol{B}^{-1}\boldsymbol{p}^{(i)} = -\boldsymbol{B}^{-1}\boldsymbol{B}\bar{\boldsymbol{g}}^{(i)} = -\bar{\boldsymbol{g}}^{(i)}$, and hence (A.31) becomes

$$M(\boldsymbol{p}^{(i)}) = \boldsymbol{p}^{(i)\top}\boldsymbol{g}^{(i+1)} - \tfrac{1}{2}\boldsymbol{p}^{(i)\top}\bar{\boldsymbol{g}}^{(i)}. \tag{A.33}$$

Similarly, we have

$$D^{(i)}(\boldsymbol{\alpha}^{(i)}) = -\tfrac{1}{2}(\boldsymbol{G}^{(i)}\boldsymbol{\alpha}^{(i)})^\top \boldsymbol{B}(\boldsymbol{G}^{(i)}\boldsymbol{\alpha}^{(i)}) = \tfrac{1}{2}\boldsymbol{p}^{(i)\top}\bar{\boldsymbol{g}}^{(i)}. \tag{A.34}$$

From (A.14) and (A.30) it follows that

$$\partial\bar{D}^{(i+1)}(0) = (\bar{\boldsymbol{g}}^{(i)} - \boldsymbol{g}^{(i+1)})^\top \boldsymbol{B}\bar{\boldsymbol{g}}^{(i)} = (\boldsymbol{g}^{(i+1)} - \bar{\boldsymbol{g}}^{(i)})^\top \boldsymbol{p}^{(i)}, \tag{A.35}$$

where $\boldsymbol{g}^{(i+1)}$ is a violating subgradient chosen via (A.3), and hence coincides with (A.32). Using (A.33) – (A.35), we obtain

$$M(\boldsymbol{p}^{(i)}) - D^{(i)}(\boldsymbol{\alpha}^{(i)}) = \left(\boldsymbol{g}^{(i+1)} - \bar{\boldsymbol{g}}^{(i)}\right)^\top \boldsymbol{p}^{(i)} = \partial\bar{D}^{(i+1)}(0). \tag{A.36}$$

Together with definition (A.19) of $\epsilon^{(i)}$, (A.36) implies that

$$\begin{aligned}\epsilon^{(i)} &= \min_{j\le i} M(\boldsymbol{p}^{(j)}) - D^{(i)}\left(\boldsymbol{\alpha}^{(i)}\right)\\ &\le M(\boldsymbol{p}^{(i)}) - D^{(i)}(\boldsymbol{\alpha}^{(i)}) = \partial\bar{D}^{(i+1)}(0).\end{aligned}$$

■

**Lemma A.2.2** *Let $f : [0,1] \to \mathbb{R}$ be a concave quadratic function with $f(0) = 0$, $\partial f(0) \in [0,a]$, and $\partial f^2(x) \ge -a$ for some $a \ge 0$. Then $\max_{x\in[0,1]} f(x) \ge \frac{(\partial f(0))^2}{2a}$.*

**Proof** Using a second-order Taylor expansion around 0, we have $f(x) \ge \partial f(0)x - \frac{a}{2}x^2$. $x^* = \partial f(0)/a$ is the unconstrained maximum of the lower bound. Since $\partial f(0) \in [0,a]$, we have $x^* \in [0,1]$. Plugging $x^*$ into the lower bound yields $(\partial f(0))^2/(2a)$. ■

**Theorem A.2.3** *Assume that at $\boldsymbol{w}$ the convex objective function $J : \mathbb{R}^d \to \mathbb{R}$ has bounded subgradient: $\|\partial J(\boldsymbol{w})\| \leq G$, and that the approximation $\boldsymbol{B}$ to the inverse Hessian has bounded eigenvalues: $\boldsymbol{B} \preceq H$. Then*

$$\epsilon^{(i)} - \epsilon^{(i+1)} \geq \frac{(\epsilon^{(i)})^2}{8G^2H}.$$

**Proof** Recall that we constrain the form of feasible dual solutions for $D^{(i+1)}(\boldsymbol{\alpha})$ as in (A.11). Instead of $D^{(i+1)}(\boldsymbol{\alpha})$, we thus work with the one-dimensional concave quadratic function $\bar{D}^{(i+1)}(\mu)$ (A.12). It is obvious that $\begin{bmatrix} \boldsymbol{\alpha}^{(i)} \\ 0 \end{bmatrix}$ is a feasible solution for $D^{(i+1)}(\boldsymbol{\alpha})$. In this case, $\bar{D}^{(i+1)}(0) = D^{(i)}(\boldsymbol{\alpha}^{(i)})$. (A.16) implies that $\bar{D}^{(i+1)}(\mu^*) = D^{(i+1)}(\boldsymbol{\alpha}^{(i+1)})$. Using the definition (A.19) of $\epsilon^{(i)}$, we thus have

$$\epsilon^{(i)} - \epsilon^{(i+1)} \geq D^{(i+1)}(\boldsymbol{\alpha}^{(i+1)}) - D^{(i)}(\boldsymbol{\alpha}^{(i)}) = \bar{D}^{(i+1)}(\mu^*) - \bar{D}^{(i+1)}(0). \tag{A.37}$$

It is easy to see from (A.37) that $\epsilon^{(i)} - \epsilon^{(i+1)}$ are upper bounds on the maximal value of the concave quadratic function $f(\mu) := \bar{D}^{(i+1)}(\mu) - \bar{D}^{(i+1)}(0)$ with $\mu \in [0, 1]$ and $f(0) = 0$. Furthermore, the definitions of $\bar{D}^{(i+1)}(\mu)$ and $f(\mu)$ imply that

$$\begin{aligned} \partial f(0) &= \partial \bar{D}^{(i+1)}(0) = (\bar{\boldsymbol{g}}^{(i)} - \boldsymbol{g}^{(i+1)})^\top \boldsymbol{B} \bar{\boldsymbol{g}}^{(i)} \quad \text{and} \\ \partial^2 f(\mu) &= \partial^2 \bar{D}^{(i+1)}(\mu) = -(\bar{\boldsymbol{g}}^{(i)} - \boldsymbol{g}^{(i+1)})^\top \boldsymbol{B} (\bar{\boldsymbol{g}}^{(i)} - \boldsymbol{g}^{(i+1)}). \end{aligned} \tag{A.38}$$

Since $\|\partial J(\boldsymbol{w})\| \leq G$ and $\bar{\boldsymbol{g}}^{(i)} \in \partial J(\boldsymbol{w})$ (A.13), we have $\|\bar{\boldsymbol{g}}^{(i)} - \boldsymbol{g}^{(i+1)}\| \leq 2G$. Our upper bound on the spectrum of $\boldsymbol{B}$ then gives $|\partial f(0)| \leq 2G^2H$ and $|\partial^2 f(\mu)| \leq 4G^2H$. Additionally, Lemma A.2.1 and the fact that $\boldsymbol{B} \succeq 0$ imply that

$$\partial f(0) = \partial \bar{D}^{(i+1)}(0) \geq 0 \quad \text{and} \quad \partial^2 f(\mu) = \partial^2 \bar{D}^{(i+1)}(\mu) \leq 0, \tag{A.39}$$

which means that

$$\partial f(0) \in [0, 2G^2H] \subset [0, 4G^2H] \quad \text{and} \quad \partial^2 f(\mu) \geq -4G^2H. \tag{A.40}$$

Invoking Lemma A.2.2, we immediately get

$$\epsilon^{(i)} - \epsilon^{(i+1)} \geq \frac{(\partial f(0))^2}{8G^2H} = \frac{(\partial \bar{D}^{(i+1)}(0))^2}{8G^2H}. \tag{A.41}$$

Since $\epsilon^{(i)} \leq \partial \bar{D}^{(i+1)}(0)$ by Lemma A.2.1, the inequality (A.41) still holds when $\partial \bar{D}^{(i+1)}(0)$ is replaced with $\epsilon^{(i)}$. ∎

(A.38) and (A.39) imply that the optimal combination coefficient $\mu^*$ (A.15) has the property

$$\mu^* = \min\left[1, \frac{\partial \bar{D}^{(i+1)}(0)}{-\partial^2 \bar{D}^{(i+1)}(\mu)}\right]. \tag{A.42}$$

Moreover, we can use (A.7) to reduce the cost of computing $\mu^*$ by setting $\boldsymbol{B}\bar{\boldsymbol{g}}^{(i)}$ in (A.15) to be $-\boldsymbol{p}^{(i)}$ (Line 7 of Algorithm 3.2), and calculate

$$\mu^* = \min\left[1, \frac{\boldsymbol{g}^{(i+1)\top}\boldsymbol{p}^{(i)} - \bar{\boldsymbol{g}}^{(i)\top}\boldsymbol{p}^{(i)}}{\boldsymbol{g}^{(i+1)\top}\boldsymbol{B}_t\boldsymbol{g}^{(i+1)} + 2\,\boldsymbol{g}^{(i+1)\top}\boldsymbol{p}^{(i)} - \bar{\boldsymbol{g}}^{(i)\top}\boldsymbol{p}^{(i)}}\right], \tag{A.43}$$

where $\boldsymbol{B}_t\boldsymbol{g}^{(i+1)}$ can be cached for the update of the primal solution at Line 9 of Algorithm 3.2.

To prove Theorem A.2.5, we use the following lemma proven by induction by Abe et al. (2001, Sublemma 5.4):

**Lemma A.2.4** *Let $\{\epsilon^{(1)}, \epsilon^{(2)}, \cdots\}$ be a sequence of non-negative numbers satisfying $\forall i \in \mathbb{N}$ the recurrence*

$$\epsilon^{(i)} - \epsilon^{(i+1)} \;\geq\; c\,(\epsilon^{(i)})^2,$$

*where $c \in \mathbb{R}_+$ is a positive constant. Then $\forall i \in \mathbb{N}$ we have*

$$\epsilon^{(i)} \;\leq\; \frac{1}{c\left(i + \frac{1}{\epsilon^{(1)}c}\right)}.$$

We now show that Algorithm 3.2 decreases $\epsilon^{(i)}$ to a pre-defined tolerance $\epsilon$ in $O(1/\epsilon)$ steps:

**Theorem A.2.5** *Under the assumptions of Theorem A.2.3, Algorithm 3.2 converges to the desired precision $\epsilon$ after*

$$1 \;\leq\; t \;\leq\; \frac{8G^2H}{\epsilon} - 4$$

*steps for any $\epsilon < 2G^2H$.*

**Proof** Theorem A.2.3 states that

$$\epsilon^{(i)} - \epsilon^{(i+1)} \;\geq\; \frac{(\epsilon^{(i)})^2}{8G^2H}, \tag{A.44}$$

where $\epsilon^{(i)}$ is non-negative $\forall i \in \mathbb{N}$ by (A.20). Applying Lemma A.2.4 we thus obtain

$$\epsilon^{(i)} \leq \frac{1}{c\left(i + \frac{1}{\epsilon^{(1)}c}\right)}, \quad \text{where} \quad c := \frac{1}{8G^2H}. \tag{A.45}$$

Our assumptions on $\|\partial J(\boldsymbol{w})\|$ and the spectrum of $\boldsymbol{B}$ imply that

$$\bar{D}^{(i+1)}(0) = (\bar{\boldsymbol{g}}^{(i)} - \boldsymbol{g}^{(i+1)})^\top \boldsymbol{B}\bar{\boldsymbol{g}}^{(i)} \leq 2G^2H. \tag{A.46}$$

Hence $\epsilon^{(i)} \leq 2G^2H$ by Lemma A.2.1. This means that (A.45) holds with $\epsilon^{(1)} = 2G^2H$. Therefore we can solve

$$\epsilon \leq \frac{1}{c\left(t + \frac{1}{\epsilon^{(1)}c}\right)} \quad \text{with} \quad c := \frac{1}{8G^2H} \text{ and } \epsilon^{(1)} := 2G^2H \tag{A.47}$$

to obtain an upper bound on $t$ such that $(\forall i \geq t)\ \epsilon^{(i)} \leq \epsilon < 2G^2H$. The solution to (A.47) is $t \leq \frac{8G^2H}{\epsilon} - 4$. ■

## A.3 Satisfiability of the Subgradient Wolfe Conditions

To formally show that there always is a positive step size that satisfies the subgradient Wolfe conditions (3.17, 3.18), we restate a result of Hiriart-Urruty and Lemaréchal (1993, Theorem VI.2.3.3) in slightly modified form:

**Lemma A.3.1** *Given two points $\boldsymbol{w} \neq \boldsymbol{w}'$ in $\mathbb{R}^d$, define $\boldsymbol{w}_\eta = \eta\boldsymbol{w}' + (1-\eta)\boldsymbol{w}$. Let $J : \mathbb{R}^d \to \mathbb{R}$ be convex. There exists $\eta \in (0,1)$ and $\tilde{\boldsymbol{g}} \in \partial J(\boldsymbol{w}_\eta)$ such that*

$$J(\boldsymbol{w}') - J(\boldsymbol{w}) = \tilde{\boldsymbol{g}}^\top(\boldsymbol{w}' - \boldsymbol{w}) \leq \hat{\boldsymbol{g}}^\top(\boldsymbol{w}' - \boldsymbol{w}),$$

*where $\hat{\boldsymbol{g}} := \arg\sup_{\boldsymbol{g}\in\partial J(\boldsymbol{w}_\eta)} \boldsymbol{g}^\top(\boldsymbol{w}' - \boldsymbol{w})$.*

**Theorem A.3.2** *Let $\boldsymbol{p}$ be a descent direction at an iterate $\boldsymbol{w}$. If $\Phi(\eta) := J(\boldsymbol{w} + \eta\boldsymbol{p})$ is bounded below, then there exists a step size $\eta > 0$ which satisfies the subgradient Wolfe conditions (3.17, 3.18).*

**Proof** Since $\boldsymbol{p}$ is a descent direction, the line $J(\boldsymbol{w}) + c_1\eta\sup_{\boldsymbol{g}\in\partial J(\boldsymbol{w})} \boldsymbol{g}^\top\boldsymbol{p}$ with $c_1 \in (0,1)$ must intersect $\Phi(\eta)$ at least once at some $\eta > 0$ (see Figure 2.1 for geometric intuition). Let $\eta'$ be the smallest such intersection point; then

$$J(\boldsymbol{w} + \eta'\boldsymbol{p}) = J(\boldsymbol{w}) + c_1\eta' \sup_{\boldsymbol{g}\in\partial J(\boldsymbol{w})} \boldsymbol{g}^\top\boldsymbol{p}. \tag{A.48}$$

Since $\Phi(\eta)$ is lower bounded, the sufficient decrease condition (3.17) holds for all $\eta'' \in [0, \eta']$. Setting $\boldsymbol{w}' = \boldsymbol{w} + \eta'\boldsymbol{p}$ in Lemma A.3.1 implies that there exists an $\eta'' \in (0, \eta')$ such that

$$J(\boldsymbol{w} + \eta'\boldsymbol{p}) - J(\boldsymbol{w}) \leq \eta' \sup_{\boldsymbol{g} \in \partial J(\boldsymbol{w}+\eta''\boldsymbol{p})} \boldsymbol{g}^\top \boldsymbol{p}. \tag{A.49}$$

Plugging (A.48) into (A.49) and simplifying it yields

$$c_1 \sup_{\boldsymbol{g} \in \partial J(\boldsymbol{w})} \boldsymbol{g}^\top \boldsymbol{p} \leq \sup_{\boldsymbol{g} \in \partial J(\boldsymbol{w}+\eta''\boldsymbol{p})} \boldsymbol{g}^\top \boldsymbol{p}. \tag{A.50}$$

Since $\boldsymbol{p}$ is a descent direction, $\sup_{\boldsymbol{g} \in \partial J(\boldsymbol{w})} \boldsymbol{g}^\top \boldsymbol{p} < 0$, and thus (A.50) also holds when $c_1$ is replaced by $c_2 \in (c_1, 1)$. ■

## A.4 Global Convergence of SubBFGS

There are technical difficulties in extending the classical BFGS convergence proof to the nonsmooth case. This route was taken by Andrew and Gao (2007), which unfortunately left their proof critically flawed: In a key step (Andrew and Gao, 2007, Equation 7) they seek to establish the non-negativity of the directional derivative $f'(\bar{x}; \bar{q})$ of a convex function $f$ at a point $\bar{x}$ in the direction $\bar{q}$, where $\bar{x}$ and $\bar{q}$ are the limit points of convergent sequences $\{x^k\}$ and $\{\hat{q}^k\}_\kappa$, respectively. They do so by taking the limit for $k \in \kappa$ of

$$f'(x^k + \tilde{\alpha}^k \hat{q}^k; \hat{q}^k) > \gamma f'(x^k; \hat{q}^k), \text{ where } \{\tilde{\alpha}^k\} \to 0 \text{ and } \gamma \in (0, 1), \tag{A.51}$$

which leads them to claim that

$$f'(\bar{x}; \bar{q}) \geq \gamma f'(\bar{x}; \bar{q}), \tag{A.52}$$

which would imply $f'(\bar{x}; \bar{q}) \geq 0$ because $\gamma \in (0, 1)$. However, $f'(x^k, \hat{q}^k)$ does not necessarily converge to $f'(\bar{x}; \bar{q})$ because the directional derivative of a nonsmooth convex function is not continuous, only *upper semi-continuous* (Bertsekas, 1999, Proposition B.23). Instead of (A.52) we thus only have

$$f'(\bar{x}; \bar{q}) \geq \gamma \limsup_{k \to \infty, k \in \kappa} f'(x^k; \hat{q}^k), \tag{A.53}$$

which does not suffice to establish the desired result: $f'(\bar{x}; \bar{q}) \geq 0$. A similar mistake is also found in the reasoning of Andrew and Gao (2007) just after Equation 7.

Instead of this flawed approach, we use the technique introduced by Birge et al. (1998) to prove the global convergence of subBFGS (Algorithm 3.1) in objective func-

**Algorithm A.1** Algorithm 1 of Birge et al. (1998)

```
1: Initialize: $t := 0$ and $w_0$
2: while not converged do
3:     Find $w_{t+1}$ that obeys
           $J(w_{t+1}) \le J(w_t) - a_t \|g_{\epsilon'_t}\|^2 + \epsilon_t$        (A.54)
           where $g_{\epsilon'_t} \in \partial_{\epsilon'_t} J(w_{t+1}),\ a_t > 0,\ \epsilon_t, \epsilon'_t \ge 0$.
4:     $t := t + 1$
5: end while
```

tion value, *i.e.*, $J(\boldsymbol{w}_t) \to \inf_{\boldsymbol{w}} J(\boldsymbol{w})$, provided that the spectrum of BFGS' inverse Hessian approximation $\boldsymbol{B}_t$ is bounded from above and below for all $t$, and the step size $\eta_t$ (obtained at Line 9) is not summable: $\sum_{t=0}^{\infty} \eta_t = \infty$.

Birge et al. (1998) provide a unified framework for convergence analysis of optimization algorithms for nonsmooth convex optimization, based on the notion of $\epsilon$-*subgradients*. Formally, $\boldsymbol{g}$ is called an $\epsilon$-subgradient of $J$ at $\boldsymbol{w}$ iff (Hiriart-Urruty and Lemaréchal, 1993, Definition XI.1.1.1)

$$(\forall \boldsymbol{w}')\ \ J(\boldsymbol{w}') \ \ge\ J(\boldsymbol{w}) + (\boldsymbol{w}' - \boldsymbol{w})^{\top} \boldsymbol{g} - \epsilon, \quad \text{where } \ \epsilon \ge 0. \tag{A.55}$$

The set of all $\epsilon$-subgradients at a point $\boldsymbol{w}$ is called the $\epsilon$-subdifferential, and denoted $\partial_\epsilon J(\boldsymbol{w})$. From the definition of subgradient (1.10), it is easy to see that $\partial J(\boldsymbol{w}) = \partial_0 J(\boldsymbol{w}) \subseteq \partial_\epsilon J(\boldsymbol{w})$. Birge et al. (1998) propose an $\epsilon$-subgradient-based algorithm (Algorithm A.1) and provide sufficient conditions for its global convergence:

**Theorem A.4.1** (Birge et al., 1998, Theorem 2.1(iv), first sentence)
*Let $J : \mathbb{R}^d \to \mathbb{R} \cup \{\infty\}$ be a proper lower semi-continuous[3] extended-valued convex function, and let $\{(\epsilon_t, \epsilon'_t, a_t, \boldsymbol{w}_{t+1}, \boldsymbol{g}_{\epsilon'_t})\}$ be any sequence generated by Algorithm A.1 satisfying*

$$\sum_{t=0}^{\infty} \epsilon_t < \infty \ \ \textit{and} \ \ \sum_{t=0}^{\infty} a_t = \infty. \tag{A.56}$$

*If $\epsilon'_t \to 0$, and there exists a positive number $\beta > 0$ such that, for all large $t$,*

$$\beta \, \|\boldsymbol{w}_{t+1} - \boldsymbol{w}_t\| \ \le\ a_t \|\boldsymbol{g}_{\epsilon'_t}\|, \tag{A.57}$$

*then $J(\boldsymbol{w}_t) \to \inf_{\boldsymbol{w}} J(\boldsymbol{w})$.*

[3]This means that there exists at least one $\boldsymbol{w} \in \mathbb{R}^d$ such that $J(\boldsymbol{w}) < \infty$, and that for all $\boldsymbol{w} \in \mathbb{R}^d$, $J(\boldsymbol{w}) > -\infty$ and $J(\boldsymbol{w}) \le \liminf_{t\to\infty} J(\boldsymbol{w}_t)$ for any sequence $\{\boldsymbol{w}_t\}$ converging to $\boldsymbol{w}$. All objective functions considered in this paper fulfill these conditions.

We will use this result to establish the global convergence of subBFGS in Theorem A.4.3. Towards this end, we first show that subBFGS is a special case of Algorithm A.1:

**Lemma A.4.2** *Let $\boldsymbol{p}_t = -\boldsymbol{B}_t\bar{\boldsymbol{g}}_t$ be the descent direction produced by Algorithm 3.2 at a non-optimal iterate $\boldsymbol{w}_t$, where $\boldsymbol{B}_t \succeq h > 0$ and $\bar{\boldsymbol{g}}_t \in \partial J(\boldsymbol{w}_t)$, and let $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \eta_t\boldsymbol{p}_t$, where $\eta_t > 0$ satisfies sufficient decrease (3.17) with free parameter $c_1 \in (0,1)$. Then $\boldsymbol{w}_{t+1}$ obeys (A.54) of Algorithm A.1 for $a_t := \frac{c_1\eta_t h}{2}$, $\epsilon_t = 0$, and $\epsilon'_t := \eta_t(1-\frac{c_1}{2})\,\bar{\boldsymbol{g}}_t^\top\boldsymbol{B}_t\bar{\boldsymbol{g}}_t$.*

**Proof** Our sufficient decrease condition (3.17) and Corollary A.1.2 imply that

$$J(\boldsymbol{w}_{t+1}) \;\le\; J(\boldsymbol{w}_t) - \frac{c_1\eta_t}{2}\,\bar{\boldsymbol{g}}_t^\top\boldsymbol{B}_t\bar{\boldsymbol{g}}_t \tag{A.58}$$

$$\le\; J(\boldsymbol{w}_t) - a_t\|\bar{\boldsymbol{g}}_t\|^2, \;\text{ where }\; a_t := \frac{c_1\eta_t h}{2}. \tag{A.59}$$

What is left to prove is that $\bar{\boldsymbol{g}}_t \in \partial_{\epsilon'_t} J(\boldsymbol{w}_{t+1})$ for an $\epsilon'_t \ge 0$. Using $\bar{\boldsymbol{g}}_t \in \partial J(\boldsymbol{w}_t)$ and the definition (1.10) of subgradient, we have

$$\begin{aligned}(\forall \boldsymbol{w})\;\; J(\boldsymbol{w}) &\ge J(\boldsymbol{w}_t) + (\boldsymbol{w}-\boldsymbol{w}_t)^\top\bar{\boldsymbol{g}}_t \\ &= J(\boldsymbol{w}_{t+1}) + (\boldsymbol{w}-\boldsymbol{w}_{t+1})^\top\bar{\boldsymbol{g}}_t + J(\boldsymbol{w}_t) - J(\boldsymbol{w}_{t+1}) + (\boldsymbol{w}_{t+1}-\boldsymbol{w}_t)^\top\bar{\boldsymbol{g}}_t\,.\end{aligned} \tag{A.60}$$

Using $\boldsymbol{w}_{t+1} - \boldsymbol{w}_t = -\eta_t\boldsymbol{B}_t\bar{\boldsymbol{g}}_t$ and (A.58) gives

$$\begin{aligned}(\forall \boldsymbol{w})\;\; J(\boldsymbol{w}) &\ge J(\boldsymbol{w}_{t+1}) + (\boldsymbol{w}-\boldsymbol{w}_{t+1})^\top\bar{\boldsymbol{g}}_t + \frac{c_1\eta_t}{2}\,\bar{\boldsymbol{g}}_t^\top\boldsymbol{B}_t\bar{\boldsymbol{g}}_t - \eta_t\,\bar{\boldsymbol{g}}_t^\top\boldsymbol{B}_t\bar{\boldsymbol{g}}_t \\ &= J(\boldsymbol{w}_{t+1}) + (\boldsymbol{w}-\boldsymbol{w}_{t+1})^\top\bar{\boldsymbol{g}}_t - \epsilon'_t\,,\end{aligned}$$

where $\epsilon'_t := \eta_t(1-\frac{c_1}{2})\,\bar{\boldsymbol{g}}_t^\top\boldsymbol{B}_t\bar{\boldsymbol{g}}_t$. Since $\eta_t > 0$, $c_1 < 1$, and $\boldsymbol{B}_t \succeq h > 0$, $\epsilon'_t$ is non-negative. By the definition (A.55) of $\epsilon$-subgradient, $\bar{\boldsymbol{g}}_t \in \partial_{\epsilon'_t} J(\boldsymbol{w}_{t+1})$. ■

**Theorem A.4.3** *Let $J : \mathbb{R}^d \to \mathbb{R}\cup\{\infty\}$ be a proper lower semi-continuous[3] extended-valued convex function. Algorithm 3.1 with a line search that satisfies the sufficient decrease condition (3.17) with $c_1 \in (0,1)$ converges globally to the minimal value of $J$, provided that:*

1. *the spectrum of its approximation to the inverse Hessian is bounded above and below: $\exists\,(h, H : 0 < h \le H < \infty) : (\forall t)\; h \preceq \boldsymbol{B}_t \preceq H$*
2. *the step size $\eta_t > 0$ satisfies $\sum_{t=0}^{\infty}\eta_t = \infty$, and*
3. *the direction-finding tolerance $\epsilon$ for Algorithm 3.2 satisfies (A.29).*

**Proof** We have already shown in Lemma A.4.2 that subBFGS is a special case of Algorithm A.1. Thus if we can show that the technical conditions of Theorem A.4.1 are met, it directly establishes the global convergence of subBFGS.

Recall that for subBFGS $a_t := \frac{c_1 \eta_t h}{2}$, $\epsilon_t = 0$, $\epsilon'_t := \eta_t(1 - \frac{c_1}{2})\,\bar{\boldsymbol{g}}_t^\top \boldsymbol{B}_t \bar{\boldsymbol{g}}_t$, and $\bar{\boldsymbol{g}}_t = \boldsymbol{g}_{\epsilon'_t}$. Our assumption on $\eta_t$ implies that $\sum_{t=0}^{\infty} a_t = \frac{c_1 h}{2} \sum_{t=0}^{\infty} \eta_t = \infty$, thus establishing (A.56). We now show that $\epsilon'_t \to 0$. Under the third condition of Theorem A.4.3, it follows from the first inequality in (A.27) in Corollary A.1.2 that

$$\sup_{\boldsymbol{g} \in \partial J(\boldsymbol{w}_t)} \boldsymbol{g}^\top \boldsymbol{p}_t \;\leq\; -\tfrac{1}{2}\, \bar{\boldsymbol{g}}_t^\top \boldsymbol{B}_t \bar{\boldsymbol{g}}_t\,, \tag{A.61}$$

where $\boldsymbol{p}_t = -\boldsymbol{B}_t \bar{\boldsymbol{g}}_t$, $\bar{\boldsymbol{g}}_t \in \partial J(\boldsymbol{w}_t)$ is the search direction returned by Algorithm 3.2. Together with the sufficient decrease condition (3.17), (A.61) implies (A.58). Now use (A.58) recursively to obtain

$$J(\boldsymbol{w}_{t+1}) \;\leq\; J(\boldsymbol{w}_0) - \frac{c_1}{2} \sum_{i=0}^{t} \eta_i\, \bar{\boldsymbol{g}}_i^\top \boldsymbol{B}_i \bar{\boldsymbol{g}}_i\,. \tag{A.62}$$

Since $J$ is proper (hence bounded from below), we have

$$\sum_{t=0}^{\infty} \eta_i\, \bar{\boldsymbol{g}}_i^\top \boldsymbol{B}_i \bar{\boldsymbol{g}}_i \;=\; \frac{1}{1 - \frac{c_1}{2}} \sum_{t=0}^{\infty} \epsilon'_i \;<\; \infty\,. \tag{A.63}$$

Recall that $\epsilon'_i \geq 0$. The bounded sum of non-negative terms in (A.63) implies that the terms in the sum must converge to zero.

Finally, to show (A.57) we use $\boldsymbol{w}_{t+1} - \boldsymbol{w}_t = -\eta_t \boldsymbol{B}_t \bar{\boldsymbol{g}}_t$, the definition of the matrix norm: $\|\boldsymbol{B}\| := \max_{\boldsymbol{x} \neq 0} \frac{\|\boldsymbol{B}\boldsymbol{x}\|}{\|\boldsymbol{x}\|}$, and the upper bound on the spectrum of $\boldsymbol{B}_t$ to write:

$$\|\boldsymbol{w}_{t+1} - \boldsymbol{w}_t\| \;=\; \eta_t \|\boldsymbol{B}_t \bar{\boldsymbol{g}}_t\| \;\leq\; \eta_t \|\boldsymbol{B}_t\| \|\bar{\boldsymbol{g}}_t\| \;\leq\; \eta_t H \|\bar{\boldsymbol{g}}_t\|. \tag{A.64}$$

Recall that $\bar{\boldsymbol{g}}_t = \boldsymbol{g}_{\epsilon'_t}$ and $a_t = \frac{c_1 \eta_t h}{2}$, and multiply both sides of (A.64) by $\frac{c_1 h}{2H}$ to obtain (A.57) with $\beta := \frac{c_1 h}{2H}$. ■

## A.5 SubBFGS Converges on Various Counterexamples

We demonstrate the global convergence of subBFGS[4] with an exact line search on various counterexamples from the literature, designed to show the failure to converge of other gradient-based algorithms.

[4]We run Algorithm 3.1 with $h = 10^{-8}$ and $\epsilon = 10^{-5}$.

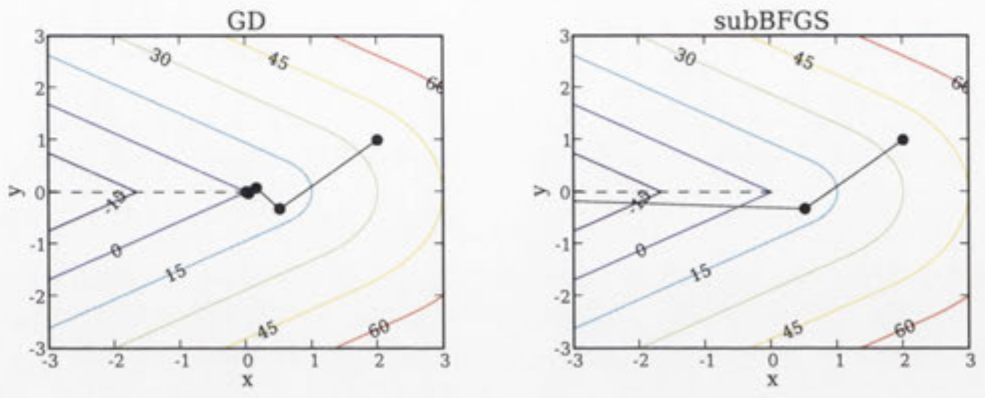


**Figure A.1:** Optimization trajectory of steepest descent (left) and subBFGS (right) on counterexample (A.65).

## A.5.1 Counterexample for Steepest Descent

The first counterexample (A.65) is given by Wolfe (1975) to show the non-convergent behaviour of the steepest descent method with an exact line search (denoted GD):

$$f(x,y) := \begin{cases} 5\sqrt{(9x^2+16y^2)} & \text{if} \quad x \geq |y|, \\ 9x + 16|y| & \text{otherwise.} \end{cases} \tag{A.65}$$

This function is subdifferentiable along $x \leq 0,\ y = 0$ (dashed line in Figure A.1); its minimal value ($-\infty$) is attained for $x = -\infty$. As can be seen in Figure A.1 (left), starting from a differentiable point $(2, 1)$, GD follows successively orthogonal directions, *i.e.*, $-\nabla f(x, y)$, and converges to the non-optimal point $(0, 0)$. As pointed out by Wolfe (1975), the failure of GD here is due to the fact that GD does not have a global view of $f$, specifically, it is because the gradient evaluated at each iterate (solid disk) is not informative about $\partial f(0, 0)$, which contains subgradients (*e.g.*, $(9, 0)$), whose negative directions point toward the minimum. SubBFGS overcomes this "short-sightedness" by incorporating into the parameter update (2.5) an estimate $\boldsymbol{B}_t$ of the inverse Hessian, whose information about the shape of $f$ prevents subBFGS from zigzagging to a non-optimal point. Figure A.1 (right) shows that subBFGS moves to the correct region ($x < 0$) at the second step. In fact, the second step of subBFGS lands exactly on the hinge $x \leq 0, y = 0$, where a subgradient pointing to the optimum is available.

## A.5.2 Counterexample for Steepest Subgradient Descent

The second counterexample (A.66), due to Hiriart-Urruty and Lemaréchal (1993, Section VIII.2.2), is a piecewise linear function which is subdifferentiable along $0 \leq y =$

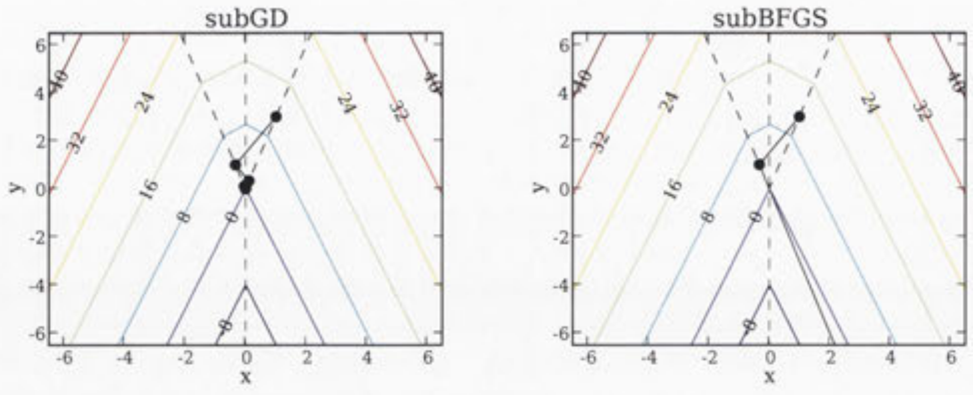


**Figure A.2:** Optimization trajectory of steepest subgradient descent (left) and subBFGS (right) on counterexample (A.66).

$\pm 3x$ and $x = 0$ (dashed lines in Figure A.2):

$$f(x, y) := \max\{-100, \ \pm 2x + 3y, \ \pm 5x + 2y\}. \tag{A.66}$$

This example shows that steepest subgradient descent with an exact line search (denoted subGD) may not converge to the optimum of a nonsmooth function. Steepest subgradient descent updates parameters along the *steepest descent* subgradient direction, which is obtained by solving the min-sup problem (3.7) with respect to the Euclidean norm. Clearly, the minimal value of $f$ $(-100)$ is attained for sufficiently negative values of $y$. However, subGD oscillates between two hinges $0 \le y = \pm 3x$, converging to the non-optimal point $(0, 0)$, as shown in Figure A.2 (left). The zigzagging optimization trajectory of subGD does not allow it to land on any informative position such as the hinge $y = 0$, where the steepest subgradient descent direction points to the desired region $(y < 0)$; Hiriart-Urruty and Lemaréchal (1993, Section VIII.2.2) provide a detailed discussion. By contrast, subBFGS moves to the $y < 0$ region at the second step (Figure A.2, right), which ends at the point $(100, -300)$ (not shown in the figure) where the minimal value of $f$ is attained .

### A.5.3 Counterexample for BFGS

The final counterexample (A.67) is given by Lewis and Overton (2008b) to show that the standard BFGS algorithm with an exact line search can break down when encountering a nonsmooth point:

$$f(x, y) := \max\{2|x| + y, \ 3y\}. \tag{A.67}$$

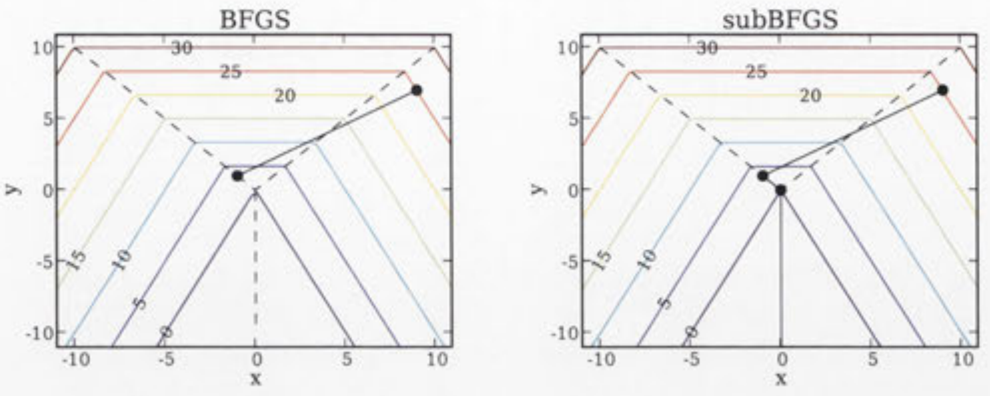


**Figure A.3:** Optimization trajectory of standard BFGS (left) and subBFGS (right) on counterexample (A.67).

This function is subdifferentiable along $x = 0,\ y \leq 0$ and $y = |x|$ (dashed lines in Figure A.3). Figure A.3 (left) shows that after the first step, BFGS lands on a nonsmooth point, where it fails to find a descent direction. This is not surprising because at a nonsmooth point $\boldsymbol{w}$ the quasi-Newton direction $\boldsymbol{p} := -\boldsymbol{B}\boldsymbol{g}$ for a given subgradient $\boldsymbol{g} \in \partial J(\boldsymbol{w})$ is not necessarily a direction of descent. SubBFGS fixes this problem by using a direction-finding procedure (Algorithm 3.2), which is guaranteed to generate a descent quasi-Newton direction. Here subBFGS converges to $f = -\infty$ in three iterations (Figure A.3, right).

# Bibliography

N. Abe, J. Takeuchi, and M. K. Warmuth. Polynomial Learnability of Stochastic Rules with Respect to the KL-Divergence and Quadratic Distance. *IEICE Transactions on Information and Systems*, 84(3):299–316, 2001.

P. K. Agarwal and M. Sharir. Davenport-schinzel sequences and their geometric applications. In J. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 1–47. North-Holland, New York, 2000.

S.-i. Amari, H. Park, and K. Fukumizu. Adaptive method of realizing natural gradient learning for multilayer perceptrons. *Neural Computation*, 12(6):1399–1409, 1998.

G. Andrew and J. Gao. Scalable training of $l_1$-regularized log-linear models. In *Proc. Intl. Conf. Machine Learning*, pages 33–40, New York, NY, USA, 2007. ACM.

J. Basch. *Kinetic Data Structures*. PhD thesis, Stanford University, June 1999.

D. P. Bertsekas. *Nonlinear Programming*. Athena Scientific, Belmont, MA, 1999.

J. R. Birge, L. Qi, and Z. Wei. A general approach to convergence properties of some methods for nonsmooth convex optimization. *Applied Mathematics and Optimization*, 38(2):141–158, 1998.

A. Bordes, L. Bottou, and P. Gallinari. Sgd-qn: Careful quasi-newton stochastic gradient descent. *Journal of Machine Learning Research*, 10:1737–1754, July 2009. URL `http://leon.bottou.org/papers/bordes-bottou-gallinari-2009`.

L. Bottou. Online algorithms and stochastic approximations. In D. Saad, editor, *Online Learning and Neural Networks*. Cambridge University Press, Cambridge, UK, 1998.

L. Bottou. Stochastic gradient descent examples on toy problems, 2009. URL `http://leon.bottou.org/projects/sgd`.

L. Bottou and Y. LeCun. Large scale online learning. In S. Thrun, L. Saul, and B. Schölkopf, editors, *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge, MA, 2004. URL `http://leon.bottou.org/papers/bottou-lecun-2004`.

L. Bottou and N. Murata. Stochastic approximations and efficient learning. In M. A. Arbib, editor, *The Handbook of Brain Theory and Neural Networks, Second edition,*. The MIT Press, Cambridge, MA, 2002. URL `http://leon.bottou.org/papers/bottou-murata-2002`.

S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, Cambridge, England, 2004.

M. Bray, E. Koller-Meier, P. Müller, L. V. Gool, and N. N. Schraudolph. 3D hand tracking by rapid stochastic gradient descent using a skinning model. In *First European Conference on Visual Media Production (CVMP)*, pages 59–68, London, 2004.

K. W. Brodlie. An assessment of two approaches to variable metric methods. *Mathematical Programming*, 12:344–355, 1977.

K. Crammer and Y. Singer. Ultraconservative online algorithms for multiclass problems. *Journal of Machine Learning Research*, 3:951–991, January 2003a.

K. Crammer and Y. Singer. A family of additive online algorithms for category ranking. *J. Mach. Learn. Res.*, 3:1025–1058, February 2003b.

J. E. Dennis and J. J. Moré. Quasi-Newton methods, motivation and theory. *SIAM Review.*, 19(1):46–89, January 1977. ISSN 0036-1445 (print), 1095-7200 (electronic).

J. E. Dennis and R. B. Schnabel. *Numerical methods for unconstrained optimization and nonlinear equations*. Classics in Applied Mathematics, 1996.

R. Fletcher. *Practical Methods of Optimization*. John Wiley and Sons, New York, 1989.

V. Franc and S. Sonnenburg. Optimized cutting plane algorithm for support vector machines. In A. McCallum and S. Roweis, editors, *ICML*, pages 320–327. Omnipress, 2008.

V. Franc and S. Sonnenburg. Optimized cutting plane algorithm for large-scale risk minimization. *Journal of Machine Learning Research*, 10:2157–2192, 2009.

M. Haarala. *Large-Scale Nonsmooth Optimization*. PhD thesis, University of Jyväskylä, 2004.

R. Hartley and F. Kahl. Optimal algorithms in multiview geometry. In *Proc. 8th Asian Conf. Computer Vision (ACCV)*, volume 1, pages 13–34, Tokyo, Japan, 2007.

J. Hershberger. Finding the upper envelope of $n$ line segments in $O(n \log n)$ time. *Information Processing Letters*, 33(4):169–174, December 1989.

J. B. Hiriart-Urruty and C. Lemaréchal. *Convex Analysis and Minimization Algorithms, I and II*, volume 305 and 306. Springer-Verlag, 1993.

L. Hirschman, A. Yeh, C. Blaschke, and A. Valencia. Overview of biocreative: critical assessment of information extraction for biology. *BMC Bioinformatics*, 6(Suppl 1): S1, 2005.

T. Joachims. Training linear SVMs in linear time. In *Proc. ACM Conf. Knowledge Discovery and Data Mining (KDD)*. ACM, 2006.

F. Kahl and R. Hartley. Multiple view geometry under the $l_\infty$-norm. *IEEE Transactions on Pattern Abnalysis and Machine Intelligence*, 30, in press 2008.

J.-D. Kim, T. Ohta, Y. Tsuruoka, Y. Tateisi, and N. Collier. Introduction to the bio-entity recognition task at JNLPBA. In *Proc. Intl. Joint Workshop on Natural Language Processing in Biomedicine and its Applications (NLPBA)*, pages 70–75, Geneva, Switzerland, 2004.

S. Kumar and M. Hebert. Discriminative fields for modeling spatial dependencies in natural images. In S. Thrun, L. Saul, and B. Schölkopf, editors, *Advances in Neural Information Processing Systems 16*, 2004.

J. D. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic modeling for segmenting and labeling sequence data. In *Proc. Intl. Conf. Machine Learning*, volume 18, pages 282–289, San Francisco, CA, 2001. Morgan Kaufmann.

Y. LeCun, L. Bottou, G. Orr, and K. M
"uller. Efficient backprop. In *Neural Networks: Tricks of the trade*. Springer, 1998.

Y. J. Lee and O. L. Mangasarian. SSVM: A smooth support vector machine for classification. *Computational optimization and Applications*, 20(1):5–22, 2001.

C. Lemarechal. Numerical experiments in nonsmooth optimization. *Progress in Nondifferentiable Optimization*, 82:61–84, 1982.

A. S. Lewis and M. L. Overton. Nonsmooth optimization via BFGS. Technical report, Optimization Online, 2008a. URL `http://www.optimization-online.org/DB_FILE/2008/12/2172.pdf`. Submitted to SIAM J. Optimization.

A. S. Lewis and M. L. Overton. Behavior of BFGS with an exact line search on nonsmooth examples. Technical report, Optimization Online, 2008b. URL `http://www.optimization-online.org/DB_FILE/2008/12/2173.pdf`. Submitted to SIAM J. Optimization.

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

L. Lukšan and J. Vlček. Globally convergent variable metric method for convex nonsmooth unconstrained minimization. *Journal of Optimization Theory and Applications*, 102(3):593–613, 1999.

F. Maes, L. Denoyer, and P. Gallinari. Xml structure mapping application to the PASCAL/INEX 2006 XML document mining track. In *Advances in XML Information Retrieval and Evaluation: Fifth Workshop of the INitiative for the Evaluation of XML Retrieval (INEX'06)*, Dagstuhl, Germany, 2007.

M. Møller. A scaled conjugate gradient algorithm for fast supervised learning. *Neural Networks*, 6(4):525–533, 1993.

A. Nedić and D. P. Bertsekas. Convergence rate of incremental subgradient algorithms. In S. Uryasev and P. M. Pardalos, editors, *Stochastic Optimization: Algorithms and Applications*, pages 263–304. Kluwer Academic Publishers, 2000.

A. Nemirovski. Prox-method with rate of convergence $o(1/t)$ for variational inequalities with lipschitz continuous monotone operators and smooth convex-concave saddle point problems. *SIAM J. on Optimization*, 15(1):229–251, 2005. ISSN 1052-6234.

Y. Nesterov. Smooth minimization of non-smooth functions. *Math. Program.*, 103(1): 127–152, 2005.

J. Nocedal. Updating quasi-newton matrices with limited storage. *Mathematics of Computation*, 35(151):773–782, 1980.

J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer Series in Operations Research. Springer, 1999.

A. I. Rauf and M. Fukushima. A globally convergent BFGS method for nonsmooth convex optimization. *Journal of Optimization Theory and Applications*, 104:2000, 1996.

H. E. Robbins and S. Monro. A stochastic approximation method. *Annals of Mathematical Statistics*, 22:400–407, 1951.

E. F. T. K. Sang and S. Buchholz. Introduction to the CoNLL-2000 shared task: Chunking. In *Proc. Conf. Computational Natural Language Learning*, pages 127–132, Lisbon, Portugal, 2000.

N. N. Schraudolph. Local gain adaptation in stochastic gradient descent. In *Proc. Intl. Conf. Artificial Neural Networks*, pages 569–574, Edinburgh, Scotland, 1999. IEE, London.

N. N. Schraudolph. Fast curvature matrix-vector products for second-order gradient descent. *Neural Computation*, 14(7):1723–1738, 2002.

N. N. Schraudolph and T. Graepel. Combining conjugate direction methods with stochastic approximation of gradients. In C. M. Bishop and B. J. Frey, editors, *Proc. 9th Intl. Workshop Artificial Intelligence and Statistics*, pages 7–13, Key West, Florida, 2003. ISBN 0-9727358-0-1.

N. N. Schraudolph, J. Yu, and D. Aberdeen. Fast online policy gradient learning with SMD gain vector adaptation. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *Advances in Neural Information Processing Systems*, volume 18, pages 1185–1192, Cambridge, MA, 2006. MIT Press.

N. N. Schraudolph, J. Yu, and S. Günter. A stochastic quasi-Newton method for online convex optimization. In M. Meila and X. Shen, editors, *Proc. 11th Intl. Conf. Artificial Intelligence and Statistics (AIstats)*, volume 2 of *Workshop and Conference Proceedings*, pages 436–443, San Juan, Puerto Rico, 2007. Journal of Machine Learning Research.

B. Settles. Biomedical named intity recognition using conditional random fields and rich feature sets. In *Proceedings of COLING 2004, International Joint Workshop On Natural Language Processing in Biomedicine and its Applications (NLPBA)*, Geneva, Switzerland, 2004.

B. Settles. ABNER: An open source tool for automatically tagging genes, proteins, and other entity names in text. *Bioinformatics*, 21(14):3191–3192, 2005.

F. Sha and F. Pereira. Shallow parsing with conditional random fields. In *Proceedings of HLT-NAACL*, pages 213–220, Edmonton, Canada, 2003. Association for Computational Linguistics.

S. Shalev-Shwartz and Y. Singer. On the equivalence of weak learnability and linear separability: New relaxations and efficient boosting algorithms. In *Proceedings of COLT*, 2008.

J. R. Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. *unpublished paper*, 1994.

A. J. Smola, S. V. N. Vishwanathan, and Q. V. Le. Bundle methods for machine learning. In D. Koller and Y. Singer, editors, *Advances in Neural Information Processing Systems 20*, Cambridge MA, 2007. MIT Press.

P. Sunehag, J. Trumpf, , S. V. N. Vishwanathan, and N. N. Schraudolph. Variable metric stochastic approximation theory. In D. van Dyk and M. Welling, editors, *Proc. 12th Intl. Conf. Artificial Intelligence and Statistics (AIstats)*, volume 5, pages 560–566. Society for Artificial Intelligence and Statistics, 2009.

B. Taskar, C. Guestrin, and D. Koller. Max-margin Markov networks. In S. Thrun, L. Saul, and B. Schölkopf, editors, *Advances in Neural Information Processing Systems 16*, pages 25–32, Cambridge, MA, 2004. MIT Press.

C.-H. Teo, S. V. N. Vishwanthan, A. J. Smola, and Q. V. Le. Bundle methods for regularized risk minimization. *Journal of Machine Learning Research*, 11:311–365, 2010.

I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun. Large margin methods for structured and interdependent output variables. *J. Mach. Learn. Res.*, 6:1453–1484, 2005.

S. V. N. Vishwanathan, N. N. Schraudolph, M. Schmidt, and K. Murphy. Accelerated training conditional random fields with stochastic gradient methods. In *Proc. Intl. Conf. Machine Learning*, pages 969–976, New York, NY, USA, 2006. ACM Press. ISBN 1-59593-383-2.

M. J. Wainwright and M. I. Jordan. Graphical models, exponential families, and variational inference. Technical Report 649, UC Berkeley, Department of Statistics, September 2003.

M. K. Warmuth, K. A. Glocer, and S. V. N. Vishwanathan. Entropy regularized LPBoost. In Y. Freund, Y. L. Györfi, and G. Turàn, editors, *Proc. Intl. Conf. Algorithmic Learning Theory*, number 5254 in Lecture Notes in Artificial Intelligence, pages 256–271, Budapest, October 2008. Springer-Verlag.

P. Wolfe. Convergence conditions for ascent methods. *SIAM Review*, 11(2):226–235, 1969.

P. Wolfe. A method of conjugate subgradients for minimizing nondifferentiable functions. *Mathematical Programming Study*, 3:145–173, 1975.

J. Yu, S. V. N. Vishwanathan, S. Günter, and N. N. Schraudolph. A quasi-Newton approach to nonsmooth convex optimization problems in machine learning. Technical Report arXiv:0804.3835, April 2008a. http://arxiv.org/pdf/0804.3835.

J. Yu, S. V. N. Vishwanathan, S. Günter, and N. N. Schraudolph. A quasi-Newton approach to nonsmooth convex optimization. In A. McCallum and S. Roweis, editors, *ICML*, pages 1216–1223. Omnipress, 2008b.

T. Zhang and F. J. Oles. Text categorization based on regularized linear classification methods. *Information Retrieval*, 4:5–31, 2001.