Optimization Methods for Large-Scale Machine Learning

Léon Bottou* Frank E. Curtis† Jorge Nocedal‡

June 23, 2018

Abstract

This paper provides a review and commentary on the past, present, and future of numerical optimization algorithms in the context of machine learning applications. Through case studies on text classification and the training of deep neural networks, we discuss how optimization problems arise in machine learning and what makes them challenging. A major theme of our study is that large-scale machine learning represents a distinctive setting in which the stochastic gradient (SG) method has traditionally played a central role while conventional gradient-based nonlinear optimization techniques typically falter. Based on this viewpoint, we present a comprehensive theory of a straightforward, yet versatile SG algorithm, discuss its practical behavior, and highlight opportunities for designing algorithms with improved performance. This leads to a discussion about the next generation of optimization methods for large-scale machine learning, including an investigation of two main streams of research on techniques that diminish noise in the stochastic directions and methods that make use of second-order derivative approximations.

Contents

*Facebook AI Research, New York, NY, USA. E-mail: leon@bottou.org

†Department of Industrial and Systems Engineering, Lehigh University, Bethlehem, PA, USA. Supported by U.S. Department of Energy grant DE–SC0010615 and U.S. National Science Foundation grant DMS–1016291. E-mail: frank.e.curtis@gmail.com

‡Department of Industrial Engineering and Management Sciences, Northwestern University, Evanston, IL, USA. Supported by Office of Naval Research grant N00014-14-1-0313 P00003 and Department of Energy grant DE-FG02-87ER25047. E-mail: j-nocedal@northwestern.edu

1 Introduction

The promise of artificial intelligence has been a topic of both public and private interest for decades. Starting in the 1950s, there has been great hope that classical artificial intelligence techniques based on logic, knowledge representation, reasoning, and planning would result in revolutionary software that could, amongst other things, understand language, control robots, and provide expert advice. Although advances based on such techniques may be in store in the future, many researchers have started to doubt these classical approaches, choosing instead to focus their efforts on the design of systems based on statistical techniques, such as in the rapidly evolving and expanding field of *machine learning*.

Machine learning and the intelligent systems that have been borne out of it—such as search engines, recommendation platforms, and speech and image recognition software—have become an indispensable part of modern society. Rooted in statistics and relying heavily on the efficiency of numerical algorithms, machine learning techniques capitalize on the world's increasingly powerful computing platforms and the availability of datasets of immense size. In addition, as the fruits of its efforts have become so easily accessible to the public through various modalities—such as *the cloud*—interest in machine learning is bound to continue its dramatic rise, yielding further societal, economic, and scientific impacts.

One of the pillars of machine learning is *mathematical optimization*, which, in this context, involves the numerical computation of parameters for a system designed to make decisions based on yet unseen data. That is, based on currently available data, these parameters are chosen to be optimal with respect to a given learning problem. The success of certain optimization methods for machine learning has inspired great numbers in various research communities to tackle even more challenging machine learning problems, and to design new methods that are more widely applicable.

The purpose of this paper is to provide a review and commentary on the past, present, and future of the use of numerical optimization algorithms in the context of machine learning applications. A major theme of this work is that large-scale machine learning represents a distinctive setting in which traditional nonlinear optimization techniques typically falter, and so should be considered secondary to alternative classes of approaches that respect the statistical nature of the underlying problem of interest.

Overall, this paper attempts to provide answers for the following questions.

1. How do optimization problems arise in machine learning applications and what makes them challenging?

2. What have been the most successful optimization methods for large-scale machine learning and why?

3. What recent advances have been made in the design of algorithms and what are open questions in this research area?

We answer the first question with the aid of two case studies. The first, a study of text classification, represents an application in which the success of machine learning has been widely recognized and celebrated. The second, a study of perceptual tasks such as speech or image recognition, represents an application in which machine learning still has had great success, but in a much more enigmatic manner that leaves many questions unanswered. These case studies also illustrate the variety of optimization problems that arise in machine learning: the first involves *convex* optimization problems—derived from the use of logistic regression or support vector machines—while the second typically involves *highly nonlinear and nonconvex* problems—derived from the use of deep neural networks.

With these case studies in hand, we turn our attention to the latter two questions on optimization algorithms, the discussions around which represent the bulk of the paper. Whereas traditional gradient-based methods may be effective for solving small-scale learning problems in which a *batch* approach may be used, in the context of large-scale machine learning it has been a *stochastic* algorithm—namely, the *stochastic gradient* method (SG) proposed by Robbins and Monro [130]—that has been the core strategy of interest. Due to this central role played by SG, we discuss its fundamental theoretical and practical properties within a few contexts of interest. We also discuss recent trends in the design of optimization methods for machine learning, organizing them according to their relationship to SG. We discuss: (*i*) noise reduction methods that attempt to borrow from the strengths of batch methods, such as their fast convergence rates and ability to exploit parallelism; (*ii*) methods that incorporate approximate second-order derivative information with the goal of dealing with nonlinearity and ill-conditioning; and (*iii*) methods for solving regularized problems designed to avoid overfitting and allow for the use of high-dimensional models. Rather than contrast SG and other methods based on the results of numerical experiments—which might bias our review toward a limited test set and implementation details—we focus our attention on fundamental computational trade-offs and theoretical properties of optimization methods.

We close the paper with a look back at what has been learned, as well as additional thoughts about the future of optimization methods for machine learning.

2 Machine Learning Case Studies

Optimization problems arise throughout machine learning. We provide two case studies that illustrate their role in the selection of prediction functions in state-of-the-art machine learning systems. We focus on cases that involve very large datasets and for which the number of model parameters to be optimized is also large. By remarking on the structure and scale of such problems, we provide a glimpse into the challenges that make them difficult to solve.

2.1 Text Classification via Convex Optimization

The assignment of natural language text to predefined classes based on their contents is one of the fundamental tasks of information management [56]. Consider, for example, the task of determining whether a text document is one that discusses politics. Educated humans can make a determination of this type unambiguously, say by observing that the document contains the names of well-known politicians. Early text classification systems attempted to consolidate knowledge from human experts by building on such observations to formally characterize the word sequences that signify a discussion about a topic of interest (e.g., politics). Unfortunately, however, concise characterizations of this type are difficult to formulate. Rules about which word sequences do or do not signify a topic need to be refined when new documents arise that cannot be classified accurately based on previously established rules. The need to coordinate such a growing collection of possibly contradictory rules limits the applicability of such systems to relatively simple tasks.

By contrast, the statistical machine learning approach begins with the collection of a sizeable set of examples $\{(x_1, y_1), \ldots, (x_n, y_n)\}$, where for each $i \in \{1, \ldots, n\}$ the vector x_i represents the *features* of a text document (e.g., the words it includes) and the scalar y_i is a *label* indicating whether the document belongs ($y_i = 1$) or not ($y_i = -1$) to a particular class (i.e., topic of interest). With such a set of examples, one can construct a classification program, defined by a *prediction function* h, and measure its performance by counting how often the program prediction $h(x_i)$ differs from the correct prediction y_i. In this manner, it seems judicious to search for a prediction function that minimizes the frequency of observed misclassifications, otherwise known as the *empirical risk* of misclassification:

$$R_n(h) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}[h(x_i) \neq y_i], \quad \text{where} \quad \mathbb{1}[A] = \begin{cases} 1 & \text{if } A \text{ is true,} \\ 0 & \text{otherwise.} \end{cases} \tag{2.1}$$

The idea of minimizing such a function gives rise to interesting conceptual issues. Consider, for example, a function that simply memorizes the examples, such as

$$h^{\text{rote}}(x) = \begin{cases} y_i & \text{if } x = x_i \text{ for some } i \in \{1, \ldots, n\}, \\ \pm 1 & \text{(arbitrarily) otherwise.} \end{cases} \tag{2.2}$$

This prediction function clearly minimizes (2.1), but it offers no performance guarantees on documents that do not appear in the examples. To avoid such rote memorization, one should aim to find a prediction function that *generalizes* the concepts that may be learned from the examples.

One way to achieve good generalized performance is to choose amongst a carefully selected class of prediction functions, perhaps satisfying certain smoothness conditions and enjoying a convenient parametric representation. How such a class of functions may be selected is not straightforward; we discuss this issue in more detail in §2.3. For now, we only mention that common practice for choosing between prediction functions belonging to a given class is to compare them using *cross-validation* procedures that involve splitting the examples into three disjoint subsets: a *training set*, a *validation set*, and a *testing set*. The process of optimizing the choice of h by minimizing R_n in (2.1) is carried out on the training set over the set of candidate prediction functions, the goal of which is to pinpoint a small subset of viable candidates. The generalized performance of each of these remaining candidates is then estimated using the validation set, the best performing of which is chosen as the selected function. The testing set is only used to estimate the generalized performance of this selected function.

Such experimental investigations have, for instance, shown that the *bag of words* approach works very well for text classification [56, 81]. In such an approach, a text document is represented by a feature vector $x \in \mathbb{R}^d$ whose components are associated with a prescribed set of vocabulary words; i.e., each nonzero component indicates that the associated word appears in the document. (The capabilities of such a representation can also be increased by augmenting the set of words with entries representing short sequences of words.) This encoding scheme leads to very sparse vectors. For instance, the canonical encoding for the standard RCV1 dataset [94] uses a vocabulary of $d = 47{,}152$ words to represent news stories that typically contain fewer than 1,000 words. Scaling techniques can be used to give more weight to distinctive words, while scaling to ensure that each document has $\|x\| = 1$ can be performed to compensate for differences in document lengths [94].

Thanks to such a high-dimensional sparse representation of documents, it has been deemed empirically sufficient to consider prediction functions of the form $h(x; w, \tau) = w^T x - \tau$. Here, $w^T x$ is a linear discriminant parameterized by $w \in \mathbb{R}^d$ and $\tau \in \mathbb{R}$ is a bias that provides a way to compromise between precision and recall.[1] The accuracy of the predictions could be determined by counting the number of times that $\mathrm{sign}(h(x; w, \tau))$ matches the correct label, i.e., 1 or -1. However, while such a prediction function may be appropriate for classifying new documents, formulating an optimization problem around it to choose the parameters (w, τ) is impractical in large-scale settings due to the combinatorial structure introduced by the sign function, which is discontinuous. Instead, one typically employs a continuous approximation through a *loss* function that measures a cost for predicting h when the true label is y; e.g., one may choose a *log-loss* function of the form $\ell(h, y) = \log(1 + \exp(-hy))$. Moreover, one obtains a class of prediction functions via the addition of a regularization term parameterized by a scalar $\lambda > 0$, leading to a convex optimization problem:

$$\min_{(w,\tau)\in\mathbb{R}^d\times\mathbb{R}} \ \frac{1}{n}\sum_{i=1}^{n} \ell(h(x_i; w, \tau), y_i) + \frac{\lambda}{2}\|w\|_2^2. \tag{2.3}$$

This problem may be solved multiple times for a given training set with various values of $\lambda > 0$, with the ultimate solution (w_*, τ_*) being the one that yields the best performance on a validation set.

Many variants of problem (2.3) have also appeared in the literature. For example, the well-known support vector machine problem [38] amounts to using the *hinge loss* $\ell(h, y) = \max(0, 1 - hy)$. Another popular feature is to use an ℓ_1-norm regularizer, namely $\lambda\|w\|_1$, which favors sparse solutions and therefore restricts attention to a subset of vocabulary words. Other more sophisticated losses can also target a specific precision/recall tradeoff or deal with hierarchies of classes; e.g., see [65, 49]. The choice of loss function is usually guided by experimentation.

In summary, both theoretical arguments [38] and experimental evidence [81] indicate that a carefully selected family of prediction functions and such high-dimensional representations of documents leads to good performance while avoiding *overfitting*—recall (2.2)—of the training set. The use of a simple surrogate, such as (2.3), facilitates experimentation and has been very successful for a variety of problems even beyond text classification. Simple surrogate models are, however, not the most effective in all applications. For example, for certain perceptual tasks, the use of deep neural networks—which lead to large-scale, highly nonlinear, and *nonconvex* optimization problems—has produced great advances that are unmatched by approaches involving simpler, convex models. We discuss such problems next.

2.2 Perceptual Tasks via Deep Neural Networks

Just like text classification tasks, perceptual tasks such as speech and image recognition are not well performed in an automated manner using computer programs based on sets of prescribed rules. For instance, the infinite diversity of writing styles easily defeats attempts to concisely specify which pixel combinations represent the digit *four*; see Figure 2.1. One may attempt to design heuristic techniques to handle such eclectic instantiations of the same object, but as previous attempts to design such techniques have often failed, computer vision researchers are increasingly embracing machine learning techniques.

During the past five years, spectacular applicative successes on perceptual problems have been achieved by machine learning techniques through the use of *deep neural networks* (DNNs). Although there are many kinds of DNNs, most recent advances have been made using essentially the same

[1] Precision and recall, defined as the probabilities $\mathbb{P}[y = 1|h(x) = 1]$ and $\mathbb{P}[h(x) = 1|y = 1]$, respectively, are convenient measures of classifier performance when the class of interest represents only a small fraction of all documents.

Fig. 2.1: No known prescribed rules express all pixel combinations that represent *four*.

types that were popular in the 1990s and almost forgotten in the 2000s [111]. What have made recent successes possible are the availability of much larger datasets and greater computational resources.

Because DNNs were initially inspired by simplified models of biological neurons [134, 135], they are often described with jargon borrowed from neuroscience. For the most part, this jargon turns into a rather effective language for describing a prediction function h whose value is computed by applying successive transformations to a given input vector $x_i \in \mathbb{R}^{d_0}$. These transformations are made in *layers*. For example, a canonical fully connected layer performs the computation

$$x_i^{(j)} = s(W_j \, x_i^{(j-1)} + b_j) \in \mathbb{R}^{d_j}, \tag{2.4}$$

where $x_i^{(0)} = x_i$, the matrix $W_j \in \mathbb{R}^{d_j \times d_{j-1}}$ and vector $b_j \in \mathbb{R}^{d_j}$ contain the jth layer parameters, and s is a component-wise nonlinear *activation* function. Popular choices for the activation function include the sigmoid function $s(x) = 1/(1+\exp(-x))$ and the hinge function $s(x) = \max\{0, x\}$ (often called a rectified linear unit (ReLU) in this context). In this manner, the ultimate output vector $x_i^{(J)}$ leads to the prediction function value $h(x_i; w)$, where the parameter vector w collects all the parameters $\{(W_1, b_1), \ldots, (W_J, b_J)\}$ of the successive layers.

Similar to (2.3), an optimization problem in this setting involves the collection of a training set $\{(x_1, y_1) \ldots (x_n, y_n)\}$ and the choice of a loss function ℓ leading to

$$\min_{w \in \mathbb{R}^d} \ \frac{1}{n} \sum_{i=1}^{n} \ell(h(x_i; w), y_i). \tag{2.5}$$

However, in contrast to (2.3), this optimization problem is highly nonlinear and nonconvex, making it intractable to solve to global optimality. That being said, machine learning experts have made great strides in the use of DNNs by computing approximate solutions by gradient-based methods. This has been made possible by the conceptually straightforward, yet crucial observation that the gradient of the objective in (2.5) with respect to the parameter vector w can be computed by the chain rule using algorithmic differentiation [71]. This differentiation technique is known in the machine learning community as *back propagation* [134, 135].

The number of layers in a DNN and the size of each layer are usually determined by performing comparative experiments and evaluating the system performance on a validation set, as in the procedure in §2.1. A contemporary fully connected neural network for speech recognition typically has five to seven layers. This amounts to tens of millions of parameters to be optimized, the training of which may require up to thousands of hours of speech data (representing hundreds of millions of training examples) and weeks of computation on a supercomputer. Figure 2.2 illustrates the word error rate gains achieved by using DNNs for acoustic modeling in three state-of-the-art speech recognition systems. These gains in accuracy are so significant that DNNs are now used in all the main commercial speech recognition products.

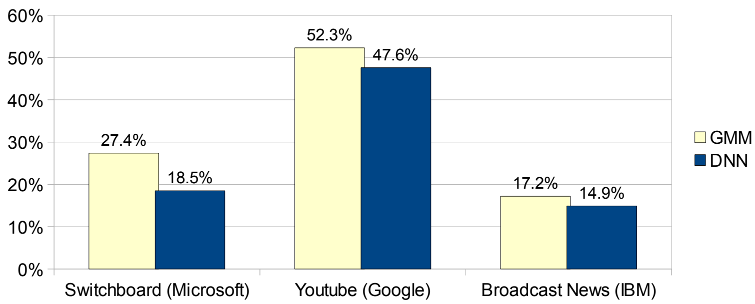

Fig. 2.2: Word error rates reported by three different research groups on three standard speech recognition benchmarks. For all three groups, deep neural networks (DNNs) significantly outperform the traditional Gaussian mixture models (GMMs) [50]. These experiments were performed between 2010 and 2012 and were instrumental in the recent revival of DNNs.

At the same time, convolutional neural networks (CNNs) have proved to be very effective for computer vision and signal processing tasks [87, 24, 88, 85]. Such a network is composed of convolutional layers, wherein the parameter matrix W_j is a circulant matrix and the input $x_i^{(j-1)}$ is intepreted as a multichannel image. The product $W_j x_i^{(j-1)}$ then computes the convolution of the image by a trainable filter while the activation function—which are piecewise linear functions as opposed to sigmoids—can perform more complex operations that may be interpreted as image rectification, contrast normalization, or subsampling. Figure 2.3 represents the architecture of the winner of the landmark 2012 ImageNet Large Scale Visual Recognition Competition (ILSVRC) [148]. The figure illustrates a CNN with five convolutional layers and three fully connected layers [85]. The input vector represents the pixel values of a 224×224 image while the output scores represent the odds that the image belongs to each of 1,000 categories. This network contains about 60 million parameters, the training of which on a few million labeled images takes a few days on a dual GPU workstation.

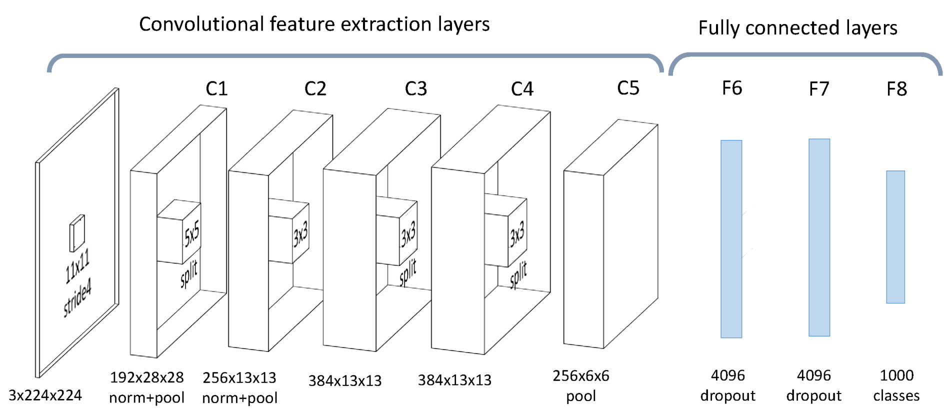

Fig. 2.3: Architecture for image recognition. The 2012 ILSVRC winner consists of eight layers [85]. Each layer performs a linear transformation (specifically, convolutions in layers C1–C5 and matrix multiplication in layers F6–F8) followed by nonlinear transformations (rectification in all layers, contrast normalization in C1–C2, and pooling in C1–C2 and C5). Regularization with dropout noise is used in layers F6–F7.

Figure 2.4 illustrates the historical error rates of the winner of the 2012 ILSVRC. In this competition, a classification is deemed successful if the correct category appeared among the top five categories returned by the system. The large performance gain achieved in 2012 was confirmed in the following years, and today CNNs are considered the tool of choice for visual object recognition [129]. They are currently deployed by numerous Internet companies for image search and face recognition.

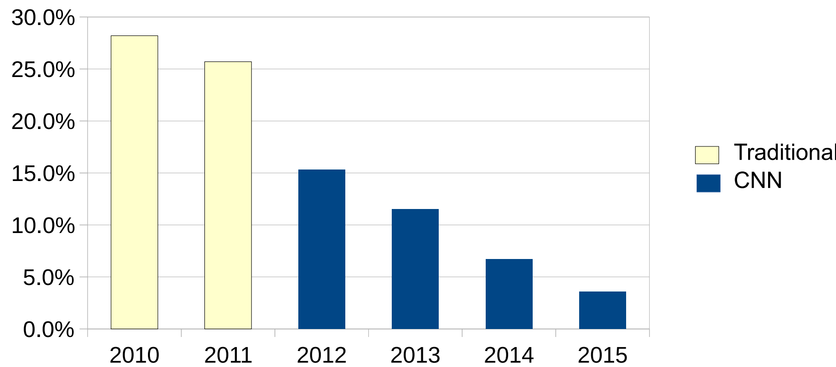

Fig. 2.4: Historical *top5* error rate of the annual winner of the ImageNet image classification challenge. A convolutional neural network (CNN) achieved a significant performance improvement over all traditional methods in 2012. The following years have cemented CNNs as the current state-of-the-art in visual object recognition [85, 129].

The successes of DNNs in modern machine learning applications are undeniable. Although the training process requires extreme skill and care—e.g., it is crucial to initialize the optimization process with a good starting point and to monitor its progress while correcting conditioning issues as they appear [89]—the mere fact that one can do anything useful with such large, highly nonlinear and nonconvex models is remarkable.

2.3 Formal Machine Learning Procedure

Through our case studies, we have illustrated how a process of machine learning leads to the selection of a prediction function h through solving an optimization problem. Moving forward, it is necessary to formalize our presentation by discussing in greater detail the principles behind the selection process, stressing the theoretical importance of *uniform laws of large numbers* as well as the practical importance of *structural risk minimization*.

For simplicity, we continue to focus on the problems that arise in the context of *supervised classification*; i.e., we focus on the optimization of prediction functions for labeling unseen data based on information contained in a set of labeled training data. Such a focus is reasonable as many unsupervised and other learning techniques reduce to optimization problems of comparable form; see, e.g., [155].

Fundamentals Our goal is to determine a prediction function $h : \mathcal{X} \to \mathcal{Y}$ from an input space $\mathcal{X}$ to an output space $\mathcal{Y}$ such that, given $x \in \mathcal{X}$, the value $h(x)$ offers an accurate prediction about the true output y. That is, our goal is to choose a prediction function that avoids rote memorization and instead generalizes the concepts that can be learned from a given set of examples. To do this, one should choose the prediction function h by attempting to minimize a risk measure over an adequately selected family of prediction functions [158], call it $\mathcal{H}$.

To formalize this idea, suppose that the examples are sampled from a joint probability distribution function $P(x, y)$ that simultaneously represents the distribution $P(x)$ of inputs as well as the conditional probability $P(y|x)$ of the label y being appropriate for an input x. (With this view, one often refers to the examples as *samples*; we use both terms throughout the rest of the paper.) Rather than one that merely minimizes the empirical risk (2.1), one should seek to find h that yields a small *expected risk* of misclassification *over all possible inputs*, i.e., an h that minimizes

$$R(h) = \mathbb{P}[h(x) \neq y] = \mathbb{E}[\mathbb{1}[h(x) \neq y]], \tag{2.6}$$

where $\mathbb{P}[A]$ and $\mathbb{E}[A]$ respectively denote the probability and expected value of A. Such a framework is *variational* since we are optimizing over a set of functions, and is *stochastic* since the objective function involves an expectation.

While one may desire to minimize the expected risk (2.6), in practice one must attempt to do so without explicit knowledge of P. Instead, the only tractable option is to construct a surrogate problem that relies solely on the examples $\{(x_i, y_i)\}_{i=1}^n$. Overall, there are two main issues that must be addressed: (i) how to choose the parameterized family of prediction functions $\mathcal{H}$ and (ii) how to determine (and find) the particular prediction function $h \in \mathcal{H}$ that is optimal.

Choice of Prediction Function Family The family of functions $\mathcal{H}$ should be determined with three *potentially competing* goals in mind. First, $\mathcal{H}$ should contain prediction functions that are able to achieve a low empirical risk over the training set, so as to avoid bias or underfitting the data. This can be achieved by selecting a rich family of functions or by using *a priori* knowledge to select a well-targeted family. Second, the gap between expected risk and empirical risk, namely, $R(h) - R_n(h)$, should be small over all $h \in \mathcal{H}$. Generally, this gap decreases when one uses more training examples, but, due to potential overfitting, it increases when one uses richer families of functions (see below). This latter fact puts the second goal at odds with the first. Third, $\mathcal{H}$ should be selected so that one can efficiently solve the corresponding optimization problem, the difficulty of which may increase when one employs a richer family of functions and/or a larger training set.

Our observation about the gap between expected and empirical risk can be understood by recalling certain *laws of large numbers*. For instance, when the expected risk represents a misclassification probability as in (2.6), the Hoeffding inequality [75] guarantees that for any fixed prediction function $h \in \mathcal{H}$, the probability to draw n examples $(x_1, y_1) \dots (x_n, y_n)$ such that

$$|R(h) - R_n(h)| \leq \sqrt{\frac{1}{2n} \log\left(\frac{2}{\eta}\right)}$$

is greater than $1-\eta$. This bound offers the intuitive explanation that the gap decreases as one uses more training examples. However, this view is insufficient for our purposes since, in the context of machine learning, the prediction function h is not fixed but depends on the training sample! Rather, h is the variable over which one is optimizing.

For this reason, one often turns to *uniform laws of large numbers* and the concept of the Vapnik-Chervonenkis (VC) dimension of $\mathcal{H}$, a measure of the *capacity* of such a family of functions [158]. For the intuition behind this concept, consider, e.g., a binary classification scheme in $\mathbb{R}^2$ where one assigns a label of 1 for points above a polynomial and -1 for points below. The set of linear polynomials has a low capacity in the sense that it is only capable of accurately classifying training points that can be separated by a line; e.g., in two variables, a linear classifier has a VC dimension of three. A set of high-degree polynomials, on the other hand, has a high capacity since it can accurately separate training points that are interspersed; the VC dimension of a polynomial of degree D in d variables is $\begin{pmatrix} d+D \\ d \end{pmatrix}$. That being said, the gap between empirical and expected risk can be larger for a set of high-degree polynomials since the high capacity allows them to overfit a given set of training data.

Mathematically, with the VC dimension measuring capacity, one can establish one of the most important results in learning theory: with $d_{\mathcal{H}}$ defined as the VC dimension of $\mathcal{H}$, one has with probability at least $1-\eta$ that

$$\sup_{h \in \mathcal{H}} |R(h) - R_n(h)| \leq \mathcal{O}\left(\sqrt{\frac{1}{2n}\log\left(\frac{2}{\eta}\right) + \frac{d_{\mathcal{H}}}{n}\log\left(\frac{n}{d_{\mathcal{H}}}\right)}\right). \tag{2.7}$$

This bound gives a more accurate picture of the dependence of the gap on the choice of $\mathcal{H}$. For example, it shows that for a fixed $d_{\mathcal{H}}$, uniform convergence is obtained by increasing the number of training points n. However, it also shows that, for a fixed n, the gap can widen for larger $d_{\mathcal{H}}$. Indeed, to maintain the same gap, one must increase n at the same rate if $d_{\mathcal{H}}$ is increased. The uniform convergence embodied in this result is crucial in machine learning since one wants to ensure that the prediction system performs well with any data provided to it. In §4.4, we employ a slight variant of this result to discuss computational trade-offs that arise in large-scale learning.[2]

Interestingly, one quantity that does not enter in (2.7) is the number of parameters that distinguish a particular member function h of the family $\mathcal{H}$. In some settings such as logistic regression, this number is essentially the same as $d_{\mathcal{H}}$, which might suggest that the task of optimizing over $h \in \mathcal{H}$ is more cumbersome as $d_{\mathcal{H}}$ increases. However, this is not always the case. Certain families of functions are amenable to minimization despite having a very large or even infinite number of parameters [156, Section 4.11]. For example, support vector machines [38] were designed to take advantage of this fact [156, Theorem 10.3].

All in all, while bounds such as (2.7) are theoretically interesting and provide useful insight, they are rarely used directly in practice since, as we have suggested in §2.1 and §2.2, it is typically easier to estimate the gap between empirical and expected risk with *cross-validation* experiments. We now present ideas underlying a practical framework that respects the trade-offs mentioned above.

Structural Risk Minimization An approach for choosing a prediction function that has proved to be widely successful in practice is *structural risk minimization* [157, 156]. Rather than choose a generic family of prediction functions—over which it would be both difficult to optimize and to estimate the gap between empirical and expected risks—one chooses a *structure*, i.e., a collection of nested function families. For instance, such a structure can be formed as a collection of subsets of a given family $\mathcal{H}$ in the following manner: given a preference function Ω, choose various values of a *hyperparameter* C, according to each of which one obtains the subset $\mathcal{H}_C := \{h \in \mathcal{H} : \Omega(h) \leq C\}$. Given a fixed number of examples, increasing C reduces the empirical risk (i.e., the minimum of

[2]We also note that considerably better bounds hold when one can collect statistics on actual examples, e.g., by determining gaps dependent on an observed variance of the risk or by considering uniform bounds restricted to families of prediction functions that achieve a risk within a certain threshold of the optimum [55, 102, 26].

$R_n(h)$ over $h \in \mathcal{H}_C$), but, after some point, it typically increases the gap between expected and empirical risks. This phenomenon is illustrated in Figure 2.5.

Other ways to introduce structures are to consider a regularized empirical risk $R_n(h) + \lambda\Omega(h)$ (an idea introduced in problem (2.3), which may be viewed as the Lagrangian for minimizing $R_n(h)$ subject to $\Omega(h) \leq C$), enlarge the dictionary in a bag-of-words representation, increase the degree of a polynomial model function, or add to the dimension of an inner layer of a DNN.

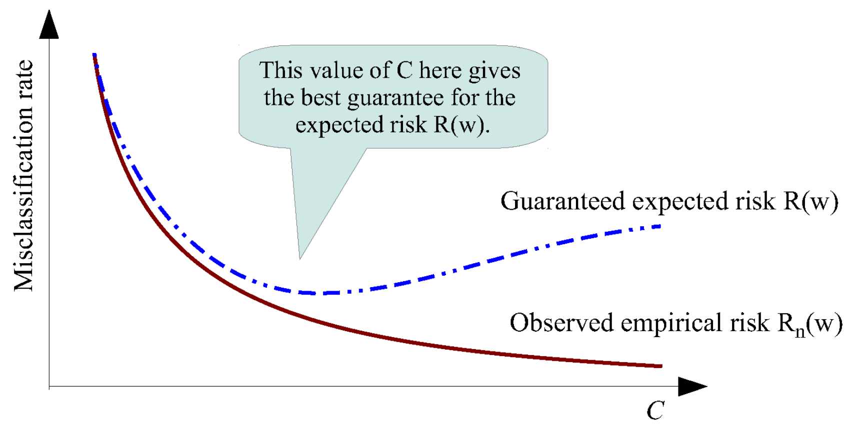

Fig. 2.5: Illustration of structural risk minimization. Given a set of n examples, a decision function family $\mathcal{H}$, and a relative preference function Ω, the figure illustrates a typical relationship between the expected and empirical risks corresponding to a prediction function obtained by an optimization algorithm that minimizes an empirical risk $R_n(h)$ subject to $\Omega(h) \leq C$. The optimal empirical risk decreases when C increases. Meanwhile, the deviation between empirical and expected risk is bounded above by a quantity—which depends on $\mathcal{H}$ and Ω—that increases with C. While not shown in the figure, the value of C that offers the best guarantee on the expected risk increases with n, i.e., the number of examples; recall (2.7).

Given such a set-up, one can avoid estimating the gap between empirical and expected risk by splitting the available data into subsets: a *training set* used to produce a subset of candidate solutions, a *validation set* used to estimate the expected risk for each such candidate, and a *testing set* used to estimate the expected risk for the candidate that is ultimately chosen. Specifically, over the training set, one minimizes an empirical risk measure R_n over $\mathcal{H}_C$ for various values of C. This results in a handful of candidate functions. The validation set is then used to estimate the expected risk corresponding to each candidate solution, after which one chooses the function yielding the lowest estimated risk value. Assuming a large enough range for C has been used, one often finds that the best solution does not correspond to the largest value of C considered; again, see Figure 2.5.

Another, albeit indirect avenue toward risk minimization is to employ an algorithm for minimizing R_n, but terminate the algorithm *early*, i.e., before an actual minimizer of R_n is found. In this manner, the role of the hyperparameter is played by the training time allowed, according to which one typically finds the relationships illustrated in Figure 2.6. Theoretical analyses related to the idea of early stopping are much more challenging than those for other forms of structural risk minimization. However, it is worthwhile to mention these effects since early stopping is a popular technique in practice, and is often *essential* due to computational budget limitations.

Overall, the structural risk minimization principle has proved useful for many applications, and

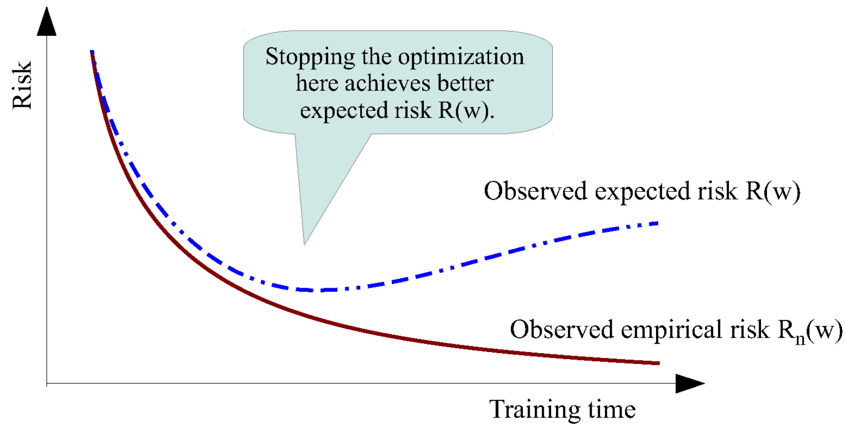

Fig. 2.6: Illustration of early stopping. Prematurely stopping the optimization of the empirical risk R_n often results in a better expected risk R. In this manner, the stopping time plays a similar role as the hyperparameter C in the illustration of structural risk minimization in Figure 2.5.

can be viewed as an alternative of the approach of employing expert human knowledge mentioned in §2.1. Rather than encoding knowledge as formal classification rules, one encodes it via preferences for certain prediction functions over others, then explores the performance of various prediction functions that have been optimized under the influence of such preferences.

3 Overview of Optimization Methods

We now turn our attention to the main focus of our study, namely, numerical algorithms for solving optimization problems that arise in large-scale machine learning. We begin by formalizing our problems of interest, which can be seen as generic statements of problems of the type described in §2 for minimizing expected and empirical risks. We then provide an overview of two main classes of optimization methods—*stochastic* and *batch*—that can be applied to solve such problems, emphasizing some of the fundamental reasons why stochastic methods have inherent advantages. We close this section with a preview of some of the advanced optimization techniques that are discussed in detail in later sections, which borrow ideas from both stochastic and batch methods.

3.1 Formal Optimization Problem Statements

As seen in §2, optimization problems in machine learning arise through the definition of prediction and loss functions that appear in measures of expected and empirical risk that one aims to minimize. Our discussions revolve around the following definitions.

Prediction and Loss Functions Rather than consider a variational optimization problem over a generic family of prediction functions, we assume that the prediction function h has a fixed form and is parameterized by a real vector $w \in \mathbb{R}^d$ over which the optimization is to be performed. Formally, for some given $h(\cdot;\cdot) : \mathbb{R}^{d_x} \times \mathbb{R}^d \to \mathbb{R}^{d_y}$, we consider the family of prediction functions

$$\mathcal{H} := \{h(\cdot; w) : w \in \mathbb{R}^d\}.$$

We aim to find the prediction function in this family that minimizes the losses incurred from inaccurate predictions. For this purpose, we assume a given loss function $\ell : \mathbb{R}^{d_y} \times \mathbb{R}^{d_y} \to \mathbb{R}$ as one that, given an input-output pair (x, y), yields the loss $\ell(h(x; w), y)$ when $h(x; w)$ and y are the predicted and true outputs, respectively.

Expected Risk Ideally, the parameter vector w is chosen to minimize the expected loss that would be incurred from *any* input-output pair. To state this idea formally, we assume that losses are measured with respect to a probability distribution $P(x, y)$ representing the true relationship between inputs and outputs. That is, we assume that the input-output space $\mathbb{R}^{d_x} \times \mathbb{R}^{d_y}$ is endowed with $P : \mathbb{R}^{d_x} \times \mathbb{R}^{d_y} \to [0, 1]$ and the objective function we wish to minimize is

$$R(w) = \int_{\mathbb{R}^{d_x} \times \mathbb{R}^{d_y}} \ell(h(x; w), y) \; dP(x, y) = \mathbb{E}[\ell(h(x; w), y)]. \tag{3.1}$$

We say that $R : \mathbb{R}^d \to \mathbb{R}$ yields the *expected risk* (i.e., expected loss) given a parameter vector w with respect to the probability distribution P.

Empirical Risk While it may be desirable to minimize (3.1), such a goal is untenable when one does not have complete information about P. Thus, in practice, one seeks the solution of a problem that involves an estimate of the expected risk R. In supervised learning, one has access (either all-at-once or incrementally) to a set of $n \in \mathbb{N}$ independently drawn input-output samples $\{(x_i, y_i)\}_{i=1}^n \subseteq \mathbb{R}^{d_x} \times \mathbb{R}^{d_y}$, with which one may define the *empirical risk* function $R_n : \mathbb{R}^d \to \mathbb{R}$ by

$$R_n(w) = \frac{1}{n} \sum_{i=1}^{n} \ell(h(x_i; w), y_i). \tag{3.2}$$

Generally speaking, minimization of R_n may be considered the practical optimization problem of interest. For now, we consider the unregularized measure (3.2), remarking that the optimization methods that we discuss in the subsequent sections can be applied readily when a smooth regularization term is included. (We leave a discussion of nonsmooth regularizers until §8.)

Note that, in §2, the functions R and R_n represented *misclassification error*; see (2.1) and (2.6). However, these new definitions of R and R_n measure the loss as determined by the function ℓ. We use these latter definitions for the rest of the paper.

Simplified Notation The expressions (3.1) and (3.2) show explicitly how the expected and empirical risks depend on the loss function, sample space or sample set, etc. However, when discussing optimization methods, we will often employ a simplified notation that also offers some avenues for generalizing certain algorithmic ideas. In particular, let us represent a sample (or set of samples) by a random seed ξ; e.g., one may imagine a realization of ξ as a single sample (x, y) from $\mathbb{R}^{d_x} \times \mathbb{R}^{d_y}$, or a realization of ξ might be a set of samples $\{(x_i, y_i)\}_{i \in \mathcal{S}}$. In addition, let us refer to the loss incurred for a given (w, ξ) as $f(w; \xi)$, i.e.,

$$f \text{ is the composition of the loss function } \ell \text{ and the prediction function } h. \tag{3.3}$$

In this manner, the expected risk for a given w is the expected value of this composite function taken with respect to the distribution of ξ:

$$\text{(Expected Risk)} \qquad R(w) = \mathbb{E}[f(w; \xi)]. \tag{3.4}$$

In a similar manner, when given a set of realizations $\{\xi_{[i]}\}_{i=1}^n$ of ξ corresponding to a sample set $\{(x_i, y_i)\}_{i=1}^n$, let us define the loss incurred by the parameter vector w with respect to the ith sample as

$$f_i(w) := f(w; \xi_{[i]}), \tag{3.5}$$

and then write the empirical risk as the average of the sample losses:

$$\text{(Empirical Risk)} \qquad R_n(w) = \frac{1}{n} \sum_{i=1}^{n} f_i(w). \tag{3.6}$$

For future reference, we use $\xi_{[i]}$ to denote the ith element of a fixed set of realizations of a random variable ξ, whereas, starting in §4, we will use ξ_k to denote the kth element of a sequence of random variables.

3.2 Stochastic vs. Batch Optimization Methods

Let us now introduce some fundamental optimization algorithms for minimizing risk. For the moment, since it is the typical setting in practice, we introduce two algorithm classes in the context of minimizing the empirical risk measure R_n in (3.6). Note, however, that much of our later discussion will focus on the performance of algorithms when considering the true measure of interest, namely, the expected risk R in (3.4).

Optimization methods for machine learning fall into two broad categories. We refer to them as *stochastic* and *batch*. The prototypical stochastic optimization method is the stochastic gradient method (SG) [130], which, in the context of minimizing R_n and with $w_1 \in \mathbb{R}^d$ given, is defined by

$$w_{k+1} \leftarrow w_k - \alpha_k \nabla f_{i_k}(w_k). \tag{3.7}$$

Here, for all $k \in \mathbb{N} := \{1, 2, \dots\}$, the index i_k (corresponding to the seed $\xi_{[i_k]}$, i.e., the sample pair (x_{i_k}, y_{i_k})) is chosen *randomly* from $\{1, \dots, n\}$ and α_k is a positive stepsize. Each iteration of this method is thus very cheap, involving only the computation of the gradient $\nabla f_{i_k}(w_k)$ corresponding to one sample. The method is notable in that the iterate sequence is not determined uniquely by the function R_n, the starting point w_1, and the sequence of stepsizes $\{\alpha_k\}$, as it would in a deterministic optimization algorithm. Rather, $\{w_k\}$ is a stochastic process whose behavior is determined by the random sequence $\{i_k\}$. Still, as we shall see in our analysis in §4, while each direction $-\nabla f_{i_k}(w_k)$ might not be one of descent from w_k (in the sense of yielding a negative directional derivative for R_n from w_k), if it is a descent direction *in expectation*, then the sequence $\{w_k\}$ can be guided toward a minimizer of R_n.

For many in the optimization research community, a *batch* approach is a more natural and well-known idea. The simplest such method in this class is the steepest descent algorithm—also referred to as the gradient, batch gradient, or full gradient method—which is defined by the iteration

$$w_{k+1} \leftarrow w_k - \alpha_k \nabla R_n(w_k) = w_k - \frac{\alpha_k}{n} \sum_{i=1}^{n} \nabla f_i(w_k). \tag{3.8}$$

Computing the step $-\alpha_k \nabla R_n(w_k)$ in such an approach is more expensive than computing the step $-\alpha_k \nabla f_{i_k}(w_k)$ in SG, though one may expect that a better step is computed when all samples are considered in an iteration.

Stochastic and batch approaches offer different trade-offs in terms of per-iteration costs and expected per-iteration improvement in minimizing empirical risk. Why, then, has SG risen to such prominence in the context of large-scale machine learning? Understanding the reasoning behind this requires careful consideration of the computational trade-offs between stochastic and batch methods, as well as a deeper look into their abilities to guarantee improvement in the underlying expected risk R. We start to investigate these topics in the next subsection.

We remark in passing that the stochastic and batch approaches mentioned here have analogues in the simulation and stochastic optimization communities, where they are referred to as *stochastic approximation* (SA) and *sample average approximation* (SAA), respectively [63].

Inset 3.1: Herbert Robbins and Stochastic Approximation

The paper by Robbins and Monro [130] represents a landmark in the history of numerical optimization methods. Together with the invention of back propagation [134, 135], it also represents one of the most notable developments in the field of machine learning. The SG method was first proposed in [130], not as a gradient method, but as a Markov chain.

Viewed more broadly, the works by Robbins and Monro [130] and Kalman [83] mark the beginning of the field of *stochastic approximation*, which studies the behavior of iterative methods that use noisy signals. The initial focus on optimization led to the study of algorithms that track the solution of the ordinary differential equation $\dot{w} = -\nabla F(w)$. Stochastic approximation theory has had a major impact in signal processing and in areas closer to the subject of this paper, such as pattern recognition [4] and neural networks [20].

After receiving his PhD, Herbert Robbins became a lecturer at New York University, where he co-authored with Richard Courant the popular book *What is Mathematics?* [39], which is still in print after more than seven decades [40]. Robbins went on to become one of the most prominent mathematicians of the second half of the twentieth century, known for his contributions to probability, algebra, and graph theory.

3.3 Motivation for Stochastic Methods

Before discussing the strengths of stochastic methods such as SG, one should not lose sight of the fact that batch approaches possess some intrinsic advantages. First, when one has reduced the stochastic problem of minimizing the expected risk R to focus exclusively on the deterministic problem of minimizing the empirical risk R_n, the use of full gradient information at each iterate opens the door for many deterministic gradient-based optimization methods. That is, in a batch approach, one has at their disposal the wealth of nonlinear optimization techniques that have been developed over the past decades, including the full gradient method (3.8), but also accelerated gradient, conjugate gradient, quasi-Newton, and inexact Newton methods [114]. (See §6 and §7 for discussion of these techniques.) Second, due to the sum structure of R_n, a batch method can easily benefit from parallelization since the bulk of the computation lies in evaluations of R_n and ∇R_n. Calculations of these quantities can even be done in a distributed manner.

Despite these advantages, there are intuitive, practical, and theoretical reasons for following a stochastic approach. Let us motivate them by contrasting the hallmark SG iteration (3.7) with the full batch gradient iteration (3.8).

Intuitive Motivation On an intuitive level, SG employs information more efficiently than a batch method. To see this, consider a situation in which a training set, call it $\mathcal{S}$, consists of ten copies of a set $\mathcal{S}_{\rm sub}$. A minimizer of empirical risk for the larger set $\mathcal{S}$ is clearly given by a minimizer for the smaller set $\mathcal{S}_{\rm sub}$, but if one were to apply a batch approach to minimize R_n over $\mathcal{S}$, then each iteration would be *ten times* more expensive than if one only had one copy of $\mathcal{S}_{\rm sub}$. On the other hand, SG performs the same computations in both scenarios, in the sense that the stochastic gradient computations involve choosing elements from $\mathcal{S}_{\rm sub}$ with the same probabilities. In reality, a training set typically does not consist of exact duplicates of sample data, but in many large-scale applications the data does involve a good deal of (approximate) redundancy. This suggests that using all of the sample data in every optimization iteration is inefficient.

A similar conclusion can be drawn by recalling the discussion in §2 related to the use of training, validation, and testing sets. If one believes that working with only, say, half of the data in the training set is sufficient to make good predictions on unseen data, then one may argue against working with the entire training set in every optimization iteration. Repeating this argument, working with only a quarter of the training set may be useful at the start, or even with only an eighth of the data, and so on. In this manner, we arrive at motivation for the idea that working with small samples, at least initially, can be quite appealing.

Practical Motivation The intuitive benefits just described have been observed repeatedly in practice, where one often finds very real advantages of SG in many applications. As an example, Figure 3.1 compares the performance of a batch L-BFGS method [97, 113] (see §6) and the SG method (3.7) with a constant stepsize (i.e., $\alpha_k = \alpha$ for all $k \in \mathbb{N}$) on a binary classification problem using a logistic loss objective function and the data from the RCV1 dataset mentioned in §2.1. The figure plots the empirical risk R_n as a function of the number of accesses of a sample from the training set, i.e., the number of evaluations of a sample gradient $\nabla f_{i_k}(w_k)$. Each set of n consecutive accesses is called an *epoch*. The batch method performs only one step per epoch while SG performs n steps per epoch. The plot shows the behavior over the first 10 epochs. The advantage of SG is striking and representative of typical behavior in practice. (One should note, however, that to obtain such efficient behavior, it was necessary to run SG repeatedly using different choices for the stepsize α until a good choice was identified for this particular problem. We discuss theoretical and practical issues related to the choice of stepsize in our analysis in §4.)

At this point, it is worthwhile to mention that the fast initial improvement achieved by SG, followed by a drastic slowdown after 1 or 2 epochs, is common in practice and is fairly well understood. An intuitive way to explain this behavior is by considering the following example due to Bertsekas [15].

Example 3.1. *Suppose that each f_i in (3.6) is a convex quadratic with minimal value at zero and minimizers $w_{i,*}$ evenly distributed in $[-1, 1]$ such that the minimizer of R_n is $w_* = 0$; see Figure 3.2. At $w_1 \ll -1$, SG will, with certainty, move to the right (toward w_*). Indeed, even if a subsequent iterate lies slightly to the right of the minimizer $w_{1,*}$ of the "leftmost" quadratic, it is likely (but not certain) that SG will continue moving to the right. However, as iterates near w_*, the algorithm enters a* region of confusion *in which there is a significant chance that a step will not move toward w_*. In this manner, progress will slow significantly. Only with more complete gradient information could the method know with certainty how to move toward w_*.*

Despite the issues illustrated by this example, we shall see in §4 that one can nevertheless ensure

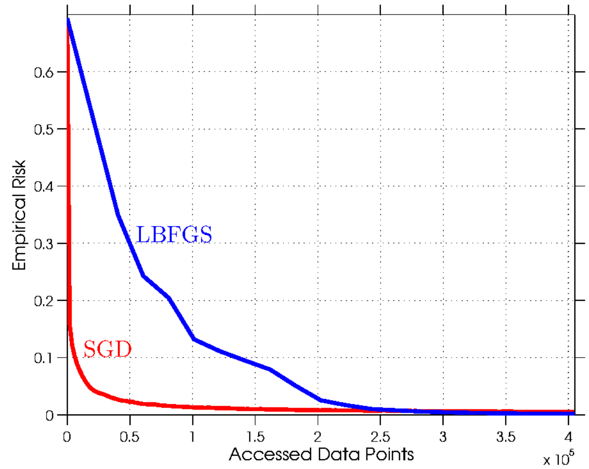

Fig. 3.1: Empirical risk R_n as a function of the number of accessed data points (ADP) for a batch L-BFGS method and the stochastic gradient (SG) method (3.7) on a binary classification problem with a logistic loss objective and the RCV1 dataset. SG was run with a fixed stepsize of $\alpha = 4$.

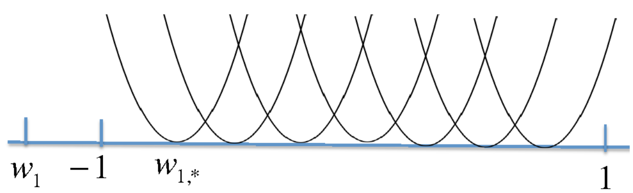

Fig. 3.2: Simple illustration to motivate the fast initial behavior of the SG method for minimizing empirical risk (3.6), where each f_i is a convex quadratic. This example is adapted from [15].

convergence by employing a sequence of diminishing stepsizes to overcome any oscillatory behavior of the algorithm.

Theoretical Motivation One can also cite theoretical arguments for a preference of SG over a batch approach. Let us give a preview of these arguments now, which are studied in more depth and further detail in §4.

- It is well known that a batch approach can minimize R_n at a fast rate; e.g., if R_n is strongly convex (see Assumption 4.5) and one applies a batch gradient method, then there exists a constant $\rho \in (0, 1)$ such that, for all $k \in \mathbb{N}$, the *training error* satisfies

$$R_n(w_k) - R_n^* \leq \mathcal{O}(\rho^k), \tag{3.9}$$

where R_n^* denotes the minimal value of R_n. The rate of convergence exhibited here is refereed to as R-linear convergence in the optimization literature [117] and geometric convergence in the machine learning research community; we shall simply refer to it as *linear convergence*. From (3.9), one can conclude that, in the worst case, the total number of iterations in which the training error can be above a given $\epsilon > 0$ is proportional to $\log(1/\epsilon)$. This means that, with a per-iteration cost proportional to n (due to the need to compute $\nabla R_n(w_k)$ for all $k \in \mathbb{N}$), the total work required to obtain ϵ-optimality for a batch gradient method is proportional to $n \log(1/\epsilon)$.

- The rate of convergence of a basic stochastic method is slower than for a batch gradient method; e.g., if R_n is strictly convex and each i_k is drawn uniformly from $\{1, \ldots, n\}$, then, for all $k \in \mathbb{N}$, the SG iterates defined by (3.7) satisfy the *sublinear convergence* property (see Theorem 4.7)

$$\mathbb{E}[R_n(w_k) - R_n^*] = \mathcal{O}(1/k). \tag{3.10}$$

  However, it is crucial to note that *neither the per-iteration cost nor the right-hand side of* (3.10) *depends on the sample set size n*. This means that the total work required to obtain ϵ-optimality for SG is proportional to $1/\epsilon$. Admittedly, this can be larger than $n \log(1/\epsilon)$ for moderate values of n and ϵ, but, as discussed in detail in §4.4, the comparison favors SG when one moves to the *big data* regime where n is large and one is merely limited by a computational time budget.

- Another important feature of SG is that, in a stochastic optimization setting, it yields the same convergence rate as in (3.10) for the error in expected risk, $R - R^*$, where R^* is the minimal value of R. Specifically, by applying the SG iteration (3.7), but with $\nabla f_{i_k}(w_k)$ replaced by $\nabla f(w_k; \xi_k)$ with each ξ_k drawn independently according to the distribution P, one finds that

$$\mathbb{E}[R(w_k) - R^*] = \mathcal{O}(1/k); \tag{3.11}$$

  again a sublinear rate, but on the expected risk. Moreover, in this context, a batch approach is not even viable without the ability to compute ∇R. Of course, this represents a different setting than one in which only a finite training set is available, but it reveals that if n is large with respect to k, then the behavior of SG in terms of minimizing the empirical risk R_n or the expected risk R is practically indistinguishable up to iteration k. This property cannot be claimed by a batch method.

In summary, there are intuitive, practical, and theoretical arguments in favor of stochastic over batch approaches in optimization methods for large-scale machine learning. For these reasons, and since SG is used so pervasively by practitioners, we frame our discussions about optimization methods in the context of their relationship with SG. We do not claim, however, that batch methods have no place in practice. For one thing, if Figure 3.1 were to consider a larger number of epochs, then one would see the batch approach eventually overtake the stochastic method and yield a lower training error. This motivates why many recently proposed methods try to combine the best properties of batch and stochastic algorithms. Moreover, the SG iteration is difficult to parallelize and requires excessive communication between nodes in a distributed computing setting, providing further impetus for the design of new and improved optimization algorithms.

3.4 Beyond SG: Noise Reduction and Second-Order Methods

Looking forward, one of the main questions being asked by researchers and practitioners alike is: what lies beyond SG that can serve as an efficient, reliable, and easy-to-use optimization method for the kinds of applications discussed in §2?

To answer this question, we depict in Figure 3.3 methods that aim to improve upon SG as lying on a two-dimensional plane. At the origin of this organizational scheme is SG, representing the base from which all other methods may be compared.

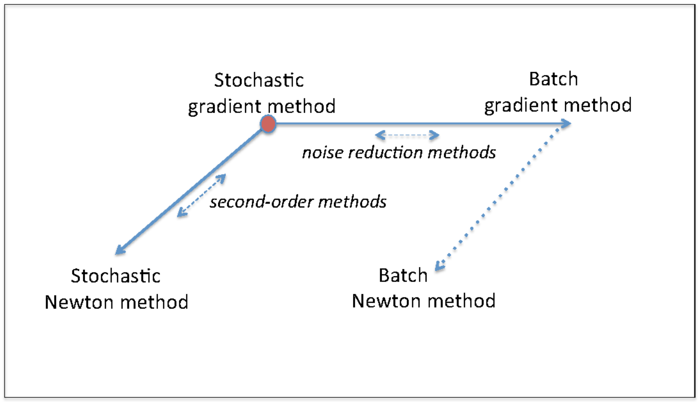

Fig. 3.3: Schematic of a two-dimensional spectrum of optimization methods for machine learning. The horizontal axis represents methods designed to control stochastic noise; the second axis, methods that deal with ill conditioning.

From the origin along the horizontal access, we place methods that are neither purely stochastic nor purely batch, but attempt to combine the best properties of both approaches. For example, observing the iteration (3.7), one quickly realizes that there is no particular reason to employ information from only one sample point per iteration. Instead, one can employ a *mini-batch* approach in which a small subset of samples, call it $\mathcal{S}_k \subseteq \{1, \ldots, n\}$, is chosen randomly in each iteration, leading to

$$w_{k+1} \leftarrow w_k - \frac{\alpha_k}{|\mathcal{S}_k|} \sum_{i \in \mathcal{S}_k} \nabla f_i(w_k). \tag{3.12}$$

Such an approach falls under the framework set out by Robbins and Monro [130], and allows some degree of parallelization to be exploited in the computation of mini-batch gradients. In addition, one often finds that, due to the reduced variance of the stochastic gradient estimates, the method is easier to tune in terms of choosing the stepsizes $\{\alpha_k\}$. Such a mini-batch SG method has been widely used in practice.

Along this horizontal axis, one finds other methods as well. In our investigation, we classify two main groups as *dynamic sample size* and *gradient aggregation* methods, both of which aim to improve the rate of convergence from sublinear to linear. These methods do not simply compute mini-batches of fixed size, nor do they compute full gradients in every iteration. Instead, they dynamically replace or incorporate new gradient information in order to construct a more reliable step with smaller variance than an SG step. For this reason, we refer to the methods along the horizontal axis as *noise reduction methods*. We discuss methods of this type in §5.

Along the second axis in Figure 3.3 are algorithms that, in a broad sense, attempt to overcome the adverse effects of high nonlinearity and ill-conditioning. For such algorithms, we use the term *second-order methods*, which encompasses a variety of strategies; see §6. We discuss well known inexact Newton and quasi-Newton methods, as well as (generalized) Gauss-Newton methods [14, 141], the natural gradient method [5], and scaled gradient iterations [152, 54].

We caution that the schematic representation of methods presented in Figure 3.3 should not be taken too literally since it is not possible to truly organize algorithms so simply, or to include all methods along only two such axes. For example, one could argue that iterate averaging methods do not neatly belong in the category of second-order methods, even though we place them there, and one could argue that gradient methods with momentum [123] or acceleration [107, 108] do belong in this category, even though we discuss them separately in §7. Nevertheless, Figure 3.3 provides a useful road map as we describe and analyze a large collection of optimization methods of various forms and characteristics. Moreover, our two-dimensional roadmap is useful in that it suggests that optimization methods do not need to exist along the coordinate axes only; e.g., a batch Newton method is placed at the lower-right corner, and one may consider various combinations of second-order and noise reduction schemes.

4 Analyses of Stochastic Gradient Methods

In this section, we provide insights into the behavior of a stochastic gradient method (SG) by establishing its convergence properties and worst-case iteration complexity bounds. A preview of such properties were given in (3.10)–(3.11), but now we prove these and other interesting results in detail, all within the context of a generalized SG algorithm. We start by analyzing our SG algorithm when it is invoked to minimize a strongly convex objective function, where it is possible to establish a global rate of convergence to the optimal objective value. This is followed by analyses when our SG algorithm is employed to minimize a generic nonconvex objective. To emphasize the generality of the results proved in this section, we remark that the objective function under consideration could be the expected risk (3.4) or empirical risk (3.6); i.e., we refer to the objective function $F : \mathbb{R}^d \to \mathbb{R}$, which represents either

$$F(w) = \begin{cases} R(w) = \mathbb{E}[f(w;\xi)] \\ \qquad\text{or} \\ R_n(w) = \dfrac{1}{n}\displaystyle\sum_{i=1}^{n} f_i(w). \end{cases} \tag{4.1}$$

Our analyses apply equally to both objectives; the only difference lies in the way that one picks the stochastic gradient estimates in the method.[3]

We define our generalized SG method as Algorithm 4.1. The algorithm merely presumes that three computational tools exist: (*i*) a mechanism for generating a realization of a random variable ξ_k (with $\{\xi_k\}$ representing a sequence of jointly independent random variables); (*ii*) given an iterate $w_k \in \mathbb{R}^d$ and the realization of ξ_k, a mechanism for computing a stochastic vector $g(w_k, \xi_k) \in \mathbb{R}^d$; and (*iii*) given an iteration number $k \in \mathbb{N}$, a mechanism for computing a scalar stepsize $\alpha_k > 0$.

[3]Picking samples uniformly from a finite training set, replacing them into the set for each iteration, corresponds to sampling from a discrete distribution giving equal weight to every sample. In this case, the SG algorithm in this section optimizes the empirical risk $F = R_n$. Alternatively, picking samples in each iteration according to the distribution P, the SG algorithm optimizes the expected risk $F = R$. One could also imagine picking samples *without replacement* until one exhausts a finite training set. In this case, the SG algorithm here can be viewed as optimizing either R_n or R, but only until the training set is exhausted. After that point, our analyses no longer apply. Generally speaking, the analyses of such *incremental algorithms* often requires specialized techniques [15, 72].

Algorithm 4.1 Stochastic Gradient (SG) Method

1: Choose an initial iterate w_1.
2: **for** $k = 1, 2, \ldots$ **do**
3: Generate a realization of the random variable ξ_k.
4: Compute a stochastic vector $g(w_k, \xi_k)$.
5: Choose a stepsize $\alpha_k > 0$.
6: Set the new iterate as $w_{k+1} \leftarrow w_k - \alpha_k g(w_k, \xi_k)$.
7: **end for**

The generality of Algorithm 4.1 can be seen in various ways. First, the value of the random variable ξ_k need only be viewed as a seed for generating a stochastic direction; as such, a realization of it may represent the choice of a single training sample as in the simple SG method stated as (3.7), or may represent a set of samples as in the mini-batch SG method (3.12). Second, $g(w_k, \xi_k)$ could represent a stochastic gradient—i.e., an unbiased estimator of $\nabla F(w_k)$, as in the classical method of Robbins and Monro [130]—or it could represent a stochastic Newton or quasi-Newton direction; see §6. That is, our analyses cover the choices

$$g(w_k, \xi_k) = \begin{cases} \nabla f(w_k; \xi_k) \\ \dfrac{1}{n_k} \displaystyle\sum_{i=1}^{n_k} \nabla f(w_k; \xi_{k,i}) \\ H_k \dfrac{1}{n_k} \displaystyle\sum_{i=1}^{n_k} \nabla f(w_k; \xi_{k,i}), \end{cases} \tag{4.2}$$

where, for all $k \in \mathbb{N}$, one has flexibility in the choice of mini-batch size n_k and symmetric positive definite scaling matrix H_k. No matter what choice is made, we shall come to see that all of our theoretical results hold as long as the expected angle between $g(w_k, \xi_k)$ and $\nabla F(w_k)$ is sufficiently positive. Third, Algorithm 4.1 allows various choices of the stepsize sequence $\{\alpha_k\}$. Our analyses focus on two choices, one involving a fixed stepsize and one involving diminishing stepsizes, as both are interesting in theory and in practice. Finally, we note that Algorithm 4.1 also covers active learning techniques in which the iterate w_k influences the sample selection.[4]

Notwithstanding all of this generality, we henceforth refer to Algorithm 4.1 as *SG*. The particular instance (3.7) will be referred to as *simple SG* or *basic SG*, whereas the instance (3.12) will be referred to as *mini-batch SG*.

Beyond our convergence and complexity analyses, a complete appreciation for the properties of SG is not possible without highlighting its theoretical advantages over batch methods in terms of computational complexity. Thus, we include in section §4.4 a discussion of the trade-offs between rate of convergence and computational effort among prototypical stochastic and batch methods for large-scale learning.

[4] We have assumed that the elements of the random variable sequence $\{\xi_k\}$ are independent in order to avoid requiring certain machinery from the analyses of stochastic processes. Viewing ξ_k as a seed instead of a sample during iteration k makes this restriction minor. However, it is worthwhile to mention that all of the results in this section still hold if, instead, $\{\xi_k\}$ forms an adapted (non-anticipating) stochastic process and expectations taken with respect to ξ_k are replaced by expectations taken with respect to the conditional distribution of ξ_k given $\{\xi_1, \ldots, \xi_{k-1}\}$.

4.1 Two Fundamental Lemmas

Our approach for establishing convergence guarantees for SG is built upon an assumption of smoothness of the objective function. (Alternative foundations are possible; see Appendix A.) This, and an assumption about the first and second moments of the stochastic vectors $\{g(w_k, \xi_k)\}$ lead to two fundamental lemmas from which all of our results will be derived.

Our first assumption is formally stated as the following. Recall that, as already mentioned in (4.1), F can stand for either expected or empirical risk.

Assumption 4.1 (Lipschitz-continuous objective gradients). *The objective function $F : \mathbb{R}^d \to \mathbb{R}$ is continuously differentiable and the gradient function of F, namely, $\nabla F : \mathbb{R}^d \to \mathbb{R}^d$, is Lipschitz continuous with Lipschitz constant $L > 0$, i.e.,*

$$\|\nabla F(w) - \nabla F(\overline{w})\|_2 \leq L\|w - \overline{w}\|_2 \;\; \textit{for all} \;\; \{w, \overline{w}\} \subset \mathbb{R}^d.$$

Intuitively, Assumption 4.1 ensures that the gradient of F does not change arbitrarily quickly with respect to the parameter vector. Such an assumption is essential for convergence analyses of most gradient-based methods; without it, the gradient would not provide a good indicator for how far to move to decrease F. An important consequence of Assumption 4.1 is that

$$F(w) \leq F(\overline{w}) + \nabla F(\overline{w})^T(w - \overline{w}) + \tfrac{1}{2}L\|w - \overline{w}\|_2^2 \;\; \text{for all} \;\; \{w, \overline{w}\} \subset \mathbb{R}^d. \tag{4.3}$$

This inequality is proved in Appendix B, but note that it also follows immediately if F is twice continuously differentiable and the Hessian function $\nabla^2 F : \mathbb{R}^d \to \mathbb{R}^{d \times d}$ satisfies $\|\nabla^2 F(w)\|_2 \leq L$ for all $w \in \mathbb{R}^d$.

Under Assumption 4.1 alone, we obtain the following lemma. In the result, we use $\mathbb{E}_{\xi_k}[\cdot]$ to denote an expected value taken with respect to the distribution of the random variable ξ_k given w_k. Therefore, $\mathbb{E}_{\xi_k}[F(w_{k+1})]$ is a meaningful quantity since w_{k+1} depends on ξ_k through the update in Step 6 of Algorithm 4.1.

Lemma 4.2. *Under Assumption 4.1, the iterates of SG (Algorithm 4.1) satisfy the following inequality for all $k \in \mathbb{N}$:*

$$\mathbb{E}_{\xi_k}[F(w_{k+1})] - F(w_k) \leq -\alpha_k \nabla F(w_k)^T \mathbb{E}_{\xi_k}[g(w_k, \xi_k)] + \tfrac{1}{2}\alpha_k^2 L \mathbb{E}_{\xi_k}[\|g(w_k, \xi_k)\|_2^2]. \tag{4.4}$$

Proof. By Assumption 4.1, the iterates generated by SG satisfy

$$\begin{aligned} F(w_{k+1}) - F(w_k) &\leq \nabla F(w_k)^T(w_{k+1} - w_k) + \tfrac{1}{2}L\|w_{k+1} - w_k\|_2^2 \\ &\leq -\alpha_k \nabla F(w_k)^T g(w_k, \xi_k) + \tfrac{1}{2}\alpha_k^2 L\|g(w_k, \xi_k)\|_2^2. \end{aligned}$$

Taking expectations in these inequalities with respect to the distribution of ξ_k, and noting that w_{k+1}—but not w_k—depends on ξ_k, we obtain the desired bound. □

This lemma shows that, regardless of how SG arrived at w_k, the expected decrease in the objective function yielded by the kth step is bounded above by a quantity involving: (i) the expected directional derivative of F at w_k along $-g(x_k, \xi_k)$ and (ii) the second moment of $g(x_k, \xi_k)$. For example, if $g(w_k, \xi_k)$ is an unbiased estimate of $\nabla F(w_k)$, then it follows from Lemma 4.2 that

$$\mathbb{E}_{\xi_k}[F(w_{k+1})] - F(w_k) \leq -\alpha_k \|\nabla F(w_k)\|_2^2 + \tfrac{1}{2}\alpha_k^2 L \mathbb{E}_{\xi_k}[\|g(w_k, \xi_k)\|_2^2]. \tag{4.5}$$

We shall see that convergence of SG is guaranteed as long as the stochastic directions and stepsizes are chosen such that the right-hand side of (4.4) is bounded above by a *deterministic* quantity that asymptotically ensures sufficient descent in F. One can ensure this in part by stating additional requirements on the first and second moments of the stochastic directions $\{g(w_k, \xi_k)\}$. In particular, in order to limit the harmful effect of the last term in (4.5), we restrict the variance of $g(w_k, \xi_k)$, i.e.,

$$\mathbb{V}_{\xi_k}[g(w_k, \xi_k)] := \mathbb{E}_{\xi_k}[\|g(w_k, \xi_k)\|_2^2] - \|\mathbb{E}_{\xi_k}[g(w_k, \xi_k)]\|_2^2 . \tag{4.6}$$

Assumption 4.3 (First and second moment limits). *The objective function and SG (Algorithm 4.1) satisfy the following:*

(a) The sequence of iterates $\{w_k\}$ is contained in an open set over which F is bounded below by a scalar $F_{\inf}$.

(b) There exist scalars $\mu_G \geq \mu > 0$ such that, for all $k \in \mathbb{N}$,

$$\nabla F(w_k)^T \mathbb{E}_{\xi_k}[g(w_k, \xi_k)] \geq \mu \|\nabla F(w_k)\|_2^2 \quad \textit{and} \tag{4.7a}$$

$$\|\mathbb{E}_{\xi_k}[g(w_k, \xi_k)]\|_2 \leq \mu_G \|\nabla F(w_k)\|_2. \tag{4.7b}$$

(c) There exist scalars $M \geq 0$ and $M_V \geq 0$ such that, for all $k \in \mathbb{N}$,

$$\mathbb{V}_{\xi_k}[g(w_k, \xi_k)] \leq M + M_V \|\nabla F(w_k)\|_2^2. \tag{4.8}$$

The first condition, Assumption 4.3(a), merely requires the objective function to be bounded below over the region explored by the algorithm. The second requirement, Assumption 4.3(b), states that, in expectation, the vector $-g(w_k, \xi_k)$ is a direction of sufficient descent for F from w_k with a norm comparable to the norm of the gradient. The properties in this requirement hold immediately with $\mu_G = \mu = 1$ if $g(w_k, \xi_k)$ is an unbiased estimate of $\nabla F(w_k)$, and are maintained if such an unbiased estimate is multiplied by a positive definite matrix H_k that is conditionally uncorrelated with $g(w_k, \xi_k)$ given w_k and whose eigenvalues lie in a fixed positive interval for all $k \in \mathbb{N}$. The third requirement, Assumption 4.3(c), states that the variance of $g(w_k, \xi_k)$ is restricted, but in a relatively minor manner. For example, if F is a convex quadratic function, then the variance is allowed to be nonzero at any stationary point for F and is allowed to grow quadratically in any direction.

All together, Assumption 4.3, combined with the definition (4.6), requires that the second moment of $g(w_k, \xi_k)$ satisfies

$$\mathbb{E}_{\xi_k}[\|g(w_k, \xi_k)\|_2^2] \leq M + M_G \|\nabla F(w_k)\|_2^2 \ \text{ with } \ M_G := M_V + \mu_G^2 \geq \mu^2 > 0. \tag{4.9}$$

In fact, all of our analyses in this section hold if this bound on the second moment were to be assumed directly. (We have stated Assumption 4.3 in the form above merely to facilitate our discussion in §5.)

The following lemma builds on Lemma 4.2 under the additional conditions now set forth in Assumption 4.3.

Lemma 4.4. *Under Assumptions 4.1 and 4.3, the iterates of SG (Algorithm 4.1) satisfy the following inequalities for all $k \in \mathbb{N}$:*

$$\mathbb{E}_{\xi_k}[F(w_{k+1})] - F(w_k) \leq -\mu \alpha_k \|\nabla F(w_k)\|_2^2 + \tfrac{1}{2}\alpha_k^2 L \mathbb{E}_{\xi_k}[\|g(w_k, \xi_k)\|_2^2] \tag{4.10a}$$

$$\leq -(\mu - \tfrac{1}{2}\alpha_k L M_G)\alpha_k \|\nabla F(w_k)\|_2^2 + \tfrac{1}{2}\alpha_k^2 L M. \tag{4.10b}$$

Proof. By Lemma 4.2 and (4.7a), it follows that

$$\begin{aligned}\mathbb{E}_{\xi_k}[F(w_{k+1})] - F(w_k) &\leq -\alpha_k \nabla F(w_k)^T \mathbb{E}_{\xi_k}[g(w_k,\xi_k)] + \tfrac{1}{2}\alpha_k^2 L \mathbb{E}_{\xi_k}[\|g(w_k,\xi_k)\|_2^2] \\ &\leq -\mu\alpha_k \|\nabla F(w_k)\|_2^2 + \tfrac{1}{2}\alpha_k^2 L \mathbb{E}_{\xi_k}[\|g(w_k,\xi_k)\|_2^2],\end{aligned}$$

which is (4.10a). Assumption 4.3, giving (4.9), then yields (4.10b). □

As mentioned, this lemma reveals that regardless of how the method arrived at the iterate w_k, the optimization process continues in a *Markovian* manner in the sense that w_{k+1} is a random variable that depends only on the iterate w_k, the seed ξ_k, and the stepsize α_k *and not on any past iterates*. This can be seen in the fact that the difference $\mathbb{E}_{\xi_k}[F(w_{k+1})] - F(w_k)$ is bounded above by a deterministic quantity. Note also that the first term in (4.10b) is strictly negative for small α_k and suggests a decrease in the objective function by a magnitude proportional to $\|\nabla F(w_k)\|_2^2$. However, the second term in (4.10b) could be large enough to allow the objective value to increase. Balancing these terms is critical in the design of SG methods.

4.2 SG for Strongly Convex Objectives

The most benign setting for analyzing the SG method is in the context of minimizing a strongly convex objective function. For the reasons described in Inset 4.2, when not considering a generic nonconvex objective F, we focus on the strongly convex case and only briefly mention the (not strongly) convex case in certain occasions.

Inset 4.2: Perspectives on SG Analyses

All of the convergence rate and complexity results presented in this paper relate to the minimizaton of *strongly convex* functions. This is in contrast with a large portion of the literature on optimization methods for machine learning, in which much effort is placed to strengthen convergence guarantees for methods applied to functions that are convex, but not strongly convex. We have made this choice for a few reasons. First, it leads to a focus on results that are relevant to actual machine learning practice, since in many situations when a convex model is employed—such as in logistic regression—it is often regularized by a strongly convex function to facilitate the solution process. Second, there exist a variety of situations in which the objective function is not globally (strongly) convex, but is so in the neighborhood of local minimizers, meaning that our results can represent the behavior of the algorithm in such regions of the search space. Third, one can argue that related results when minimizing non-strongly convex models can be derived as extensions of the results presented here [3], making our analyses a starting point for deriving a more general theory.

We have also taken a pragmatic approach in the types of convergence guarantees that we provide. In particular, in our analyses, we focus on results that reveal the properties of SG iterates *in expectation*. The stochastic approximation literature, on the other hand, often relies on martingale techniques to establish *almost sure convergence* [66, 131] under the same assumptions [21]. For our purposes, we omit these complications since, in our view, they do not provide significant additional insights into the forces driving convergence of the method.

We formalize a strong convexity assumption as the following.

Assumption 4.5 (**Strong convexity**). *The objective function $F : \mathbb{R}^d \to \mathbb{R}$ is strongly convex in that there exists a constant $c > 0$ such that*

$$F(\overline{w}) \geq F(w) + \nabla F(w)^T(\overline{w} - w) + \tfrac{1}{2}c\|\overline{w} - w\|_2^2 \ \ \textit{for all} \ \ (\overline{w}, w) \in \mathbb{R}^d \times \mathbb{R}^d. \tag{4.11}$$

Hence, F has a unique minimizer, denoted as $w_* \in \mathbb{R}^d$ with $F_* := F(w_*)$.

A useful fact from convex analysis (proved in Appendix B) is that, under Assumption 4.5, one can bound the optimality gap at a given point in terms of the squared ℓ_2-norm of the gradient of the objective at that point:

$$2c(F(w) - F_*) \leq \|\nabla F(w)\|_2^2 \ \ \text{for all} \ \ w \in \mathbb{R}^d. \tag{4.12}$$

We use this inequality in several proofs. We also observe that, from (4.3) and (4.11), the constants in Assumptions 4.1 and 4.5 must satisfy $c \leq L$.

We now state our first convergence theorem for SG, describing its behavior when minimizing a strongly convex objective function when employing a fixed stepsize. In this case, it will not be possible to prove convergence to the solution, but only to a neighborhood of the optimal value. (Intuitively, this limitation should be clear from (4.10b) since the first term on the right-hand side decreases in magnitude as the solution is approached—i.e., as $\nabla F(w_k)$ tends to zero—but the last term remains constant. Thus, after some point, a reduction in the objective cannot be expected.) We use $\mathbb{E}[\cdot]$ to denote an expected value taken with respect to the joint distribution of all random variables. For example, since w_k is completely determined by the realizations of the independent random variables $\{\xi_1, \xi_2, \ldots, \xi_{k-1}\}$, the *total expectation* of $F(w_k)$ for any $k \in \mathbb{N}$ can be taken as

$$\mathbb{E}[F(w_k)] = \mathbb{E}_{\xi_1}\mathbb{E}_{\xi_2} \ldots \mathbb{E}_{\xi_{k-1}}[F(w_k)].$$

The theorem shows that if the stepsize is not too large, then, in expectation, the sequence of function values $\{F(w_k)\}$ converges near the optimal value.

Theorem 4.6 (**Strongly Convex Objective, Fixed Stepsize**). *Under Assumptions 4.1, 4.3, and 4.5 (with $F_{\text{inf}} = F_*$), suppose that the SG method (Algorithm 4.1) is run with a fixed stepsize, $\alpha_k = \bar{\alpha}$ for all $k \in \mathbb{N}$, satisfying*

$$0 < \bar{\alpha} \leq \frac{\mu}{LM_G}. \tag{4.13}$$

Then, the expected optimality gap satisfies the following inequality for all $k \in \mathbb{N}$:

$$\begin{aligned} \mathbb{E}[F(w_k) - F_*] &\leq \frac{\bar{\alpha}LM}{2c\mu} + (1 - \bar{\alpha}c\mu)^{k-1}\left(F(w_1) - F_* - \frac{\bar{\alpha}LM}{2c\mu}\right) \\ &\xrightarrow{k\to\infty} \frac{\bar{\alpha}LM}{2c\mu}. \end{aligned} \tag{4.14}$$

Proof. Using Lemma 4.4 with (4.13) and (4.12), we have for all $k \in \mathbb{N}$ that

$$\begin{aligned} \mathbb{E}_{\xi_k}[F(w_{k+1})] - F(w_k)] &\leq -(\mu - \tfrac{1}{2}\bar{\alpha}LM_G)\bar{\alpha}\|\nabla F(w_k)\|_2^2 + \tfrac{1}{2}\bar{\alpha}^2 LM \\ &\leq -\tfrac{1}{2}\bar{\alpha}\mu\|\nabla F(w_k)\|_2^2 + \tfrac{1}{2}\bar{\alpha}^2 LM \\ &\leq -\bar{\alpha}c\mu(F(w_k) - F_*) + \tfrac{1}{2}\bar{\alpha}^2 LM. \end{aligned}$$

Subtracting F_* from both sides, taking total expectations, and rearranging, this yields

$$\mathbb{E}[F(w_{k+1}) - F_*] \leq (1 - \bar{\alpha} c\mu)\mathbb{E}[F(w_k) - F_*] + \tfrac{1}{2}\bar{\alpha}^2 LM.$$

Subtracting the constant $\bar{\alpha}LM/(2c\mu)$ from both sides, one obtains

$$\begin{aligned}\mathbb{E}[F(w_{k+1}) - F_*] - \frac{\bar{\alpha}LM}{2c\mu} &\leq (1 - \bar{\alpha}c\mu)\,\mathbb{E}[F(w_k) - F_*] + \frac{\bar{\alpha}^2 LM}{2} - \frac{\bar{\alpha}LM}{2c\mu} \\ &= (1 - \bar{\alpha}c\mu)\left(\mathbb{E}[F(w_k) - F_*] - \frac{\bar{\alpha}LM}{2c\mu}\right). \end{aligned} \tag{4.15}$$

Observe that (4.15) is a contraction inequality since, by (4.13) and (4.9),

$$0 < \bar{\alpha}c\mu \leq \frac{c\mu^2}{LM_G} \leq \frac{c\mu^2}{L\mu^2} = \frac{c}{L} \leq 1. \tag{4.16}$$

The result thus follows by applying (4.15) repeatedly through iteration $k \in \mathbb{N}$. $\square$

If $g(w_k, \xi_k)$ is an unbiased estimate of $\nabla F(w_k)$, then $\mu = 1$, and if there is no noise in $g(w_k, \xi_k)$, then we may presume that $M_G = 1$ (due to (4.9)). In this case, (4.13) reduces to $\bar{\alpha} \in (0, 1/L]$, a classical stepsize requirement of interest for a steepest descent method.

Theorem 4.6 illustrates the interplay between the stepsizes and bound on the variance of the stochastic directions. If there were no noise in the gradient computation or if noise were to decay with $\|\nabla F(w_k)\|_2^2$ (i.e., if $M = 0$ in (4.8) and (4.9)), then one can obtain linear convergence to the optimal value. This is a standard result for the full gradient method with a sufficiently small positive stepsize. On the other hand, when the gradient computation is noisy, one clearly loses this property. One can still use a fixed stepsize and be sure that the expected objective values will converge linearly to a neighborhood of the optimal value, but, after some point, the noise in the gradient estimates prevent further progress; recall Example 3.1. It is apparent from (4.14) that selecting a smaller stepsize worsens the contraction constant in the convergence rate, but allows one to arrive closer to the optimal value.

These observations provide a foundation for a strategy often employed in practice by which SG is run with a fixed stepsize, and, if progress appears to stall, a smaller stepsize is selected and the process is repeated. A straightforward instance of such an approach can be motivated with the following sketch. Suppose that $\alpha_1 \in (0, \frac{\mu}{LM_G}]$ is chosen as in (4.13) and the SG method is run with this stepsize from iteration $k_1 = 1$ until iteration k_2, where w_{k_2} is the first iterate at which the expected suboptimality gap is smaller than twice the asymptotic value in (4.14), i.e., $\mathbb{E}[F(w_{k_2}) - F_*] \leq 2F_{\alpha_1}$, where $F_\alpha := \frac{\alpha LM}{2c\mu}$. Suppose further that, at this point, the stepsize is halved and the process is repeated; see Figure 4.1. This leads to the stepsize schedule $\{\alpha_{r+1}\} = \{\alpha_1 2^{-r}\}$, index sequence $\{k_r\}$, and bound sequence $\{F_{\alpha_r}\} = \{\frac{\alpha_r LM}{2c\mu}\} \searrow 0$ such that, for all $r \in \{2, 3, \dots\}$,

$$\mathbb{E}[F(w_{k_{r+1}}) - F_*] \leq 2F_{\alpha_r} \;\; \text{where} \;\; \mathbb{E}[F(w_{k_r}) - F_*] \approx 2F_{\alpha_{r-1}} = 4F_{\alpha_r}. \tag{4.17}$$

In this manner, the expected suboptimality gap converges to zero.

However, this does *not* occur by halving the stepsize in every iteration, but only once the gap itself has been cut in half from a previous threshold. To see what is the appropriate *effective rate* of stepsize decrease, we may invoke Theorem 4.6, from which it follows that to achieve the first bound in (4.17) one needs

$$\begin{aligned}(1-\alpha_r c\mu)^{(k_{r+1}-k_r)}(4F_{\alpha_r}-F_{\alpha_r}) &\leq F_{\alpha_r} \\ \Longrightarrow k_{r+1}-k_r &\geq \frac{\log(1/3)}{\log(1-\alpha_r c\mu)} \approx \frac{\log(3)}{\alpha_r c\mu} = \mathcal{O}(2^r).\end{aligned} \tag{4.18}$$

In other words, each time the stepsize is cut in half, double the number of iterations are required. This is a *sublinear* rate of stepsize decrease—e.g., if $\{k_r\} = \{2^{r-1}\}$, then $\alpha_k = \alpha_1/k$ for all $k \in \{2^r\}$—which, from $\{F_{\alpha_r}\} = \{\frac{\alpha_r LM}{2c\mu}\}$ and (4.17), means that a sublinear convergence rate of the suboptimality gap is achieved.

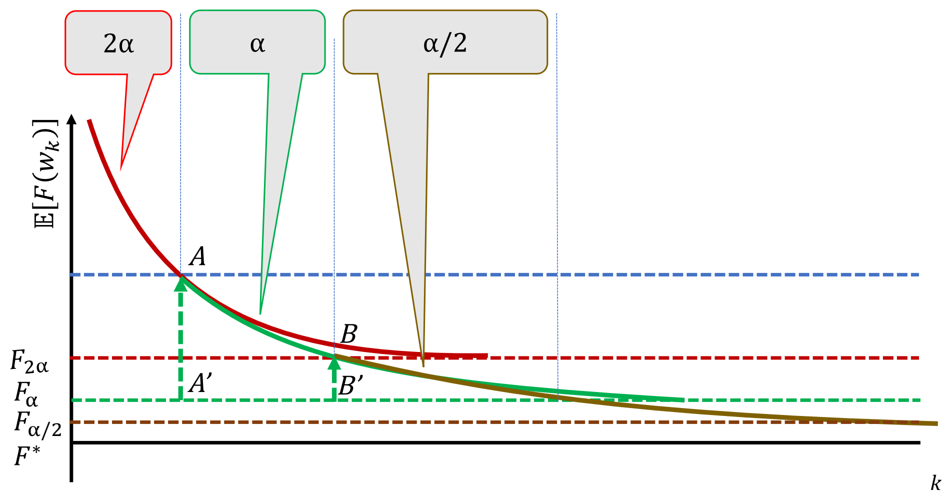

Fig. 4.1: Depiction of the strategy of halving the stepsize α when the expected suboptimality gap is smaller than twice the asymptotic limit F_α. In the figure, the segment B–B' has one third of the length of A–A'. This is the amount of decrease that must be made in the exponential term in (4.14) by raising the contraction factor to the power of the number of steps during which one maintains a given constant stepsize; see (4.18). Since the contraction factor is $(1-\alpha c\mu)$, the number of steps must be proportional to α. Therefore, whenever the stepsize is halved, one must maintain it twice as long. Overall, doubling the number of iterations halves the suboptimality gap each time, yielding an effective rate of $\mathcal{O}(1/k)$.

In fact, these conclusions can be obtained in a more rigorous manner that also allows more flexibility in the choice of stepsize sequence. The following result harks back to the seminal work of Robbins and Monro [130], where the stepsize requirement takes the form

$$\sum_{k=1}^{\infty} \alpha_k = \infty \quad \text{and} \quad \sum_{k=1}^{\infty} \alpha_k^2 < \infty. \tag{4.19}$$

Theorem 4.7 (**Strongly Convex Objective, Diminishing Stepsizes**). *Under Assumptions 4.1, 4.3, and 4.5 (with $F_{\inf} = F_*$), suppose that the SG method (Algorithm 4.1) is run with a stepsize sequence such that, for all $k \in \mathbb{N}$,*

$$\alpha_k = \frac{\beta}{\gamma+k} \ \textit{ for some } \ \beta > \frac{1}{c\mu} \ \textit{ and } \ \gamma > 0 \ \textit{ such that } \ \alpha_1 \leq \frac{\mu}{LM_G}. \tag{4.20}$$

Then, for all $k \in \mathbb{N}$, the expected optimality gap satisfies

$$\mathbb{E}[F(w_k) - F_*] \leq \frac{\nu}{\gamma + k}, \tag{4.21}$$

where

$$\nu := \max\left\{\frac{\beta^2 LM}{2(\beta c\mu - 1)}, (\gamma+1)(F(w_1) - F_*)\right\}. \tag{4.22}$$

Proof. By (4.20), the inequality $\alpha_k LM_G \leq \alpha_1 LM_G \leq \mu$ holds for all $k \in \mathbb{N}$. Hence, along with Lemma 4.4 and (4.12), one has for all $k \in \mathbb{N}$ that

$$\begin{aligned}
\mathbb{E}_{\xi_k}[F(w_{k+1})] - F(w_k) &\leq -(\mu - \tfrac{1}{2}\alpha_k LM_G)\alpha_k \|\nabla F(w_k)\|_2^2 + \tfrac{1}{2}\alpha_k^2 LM \\
&\leq -\tfrac{1}{2}\alpha_k \mu \|\nabla F(w_k)\|_2^2 + \tfrac{1}{2}\alpha_k^2 LM \\
&\leq -\alpha_k c\mu(F(w_k) - F(w_*)) + \tfrac{1}{2}\alpha_k^2 LM.
\end{aligned}$$

Subtracting F_* from both sides, taking total expectations, and rearranging, this yields

$$\mathbb{E}[F(w_{k+1}) - F_*] \leq (1 - \alpha_k c\mu)\mathbb{E}[F(w_k) - F_*] + \tfrac{1}{2}\alpha_k^2 LM. \tag{4.23}$$

We now prove (4.21) by induction. First, the definition of ν ensures that it holds for $k = 1$. Then, assuming (4.21) holds for some $k \geq 1$, it follows from (4.23) that

$$\begin{aligned}
\mathbb{E}[F(w_{k+1}) - F_*] &\leq \left(1 - \frac{\beta c\mu}{\hat{k}}\right)\frac{\nu}{\hat{k}} + \frac{\beta^2 LM}{2\hat{k}^2} \qquad (\text{with } \hat{k} := \gamma + k) \\
&= \left(\frac{\hat{k} - \beta c\mu}{\hat{k}^2}\right)\nu + \frac{\beta^2 LM}{2\hat{k}^2} \\
&= \left(\frac{\hat{k} - 1}{\hat{k}^2}\right)\nu \underbrace{- \left(\frac{\beta c\mu - 1}{\hat{k}^2}\right)\nu + \frac{\beta^2 LM}{2\hat{k}^2}}_{\text{nonpositive by the definition of } \nu} \leq \frac{\nu}{\hat{k}+1},
\end{aligned}$$

where the last inequality follows because $\hat{k}^2 \geq (\hat{k}+1)(\hat{k}-1)$. □

Let us now remark on what can be learned from Theorems 4.6 and 4.7.

Role of Strong Convexity Observe the crucial role played by the strong convexity parameter $c > 0$, the positivity of which is needed to argue that (4.15) and (4.23) contract the expected optimality gap. However, the strong convexity constant impacts the stepsizes in different ways in Theorems 4.6 and 4.7. In the case of constant stepsizes, the possible values of $\bar{\alpha}$ are constrained by the upper bound (4.13) that does not depend on c. In the case of diminishing stepsizes, the initial stepsize α_1 is subject to the same upper bound (4.20), but the stepsize parameter β must be larger than $1/(c\mu)$. This additional requirement is critical to ensure the $\mathcal{O}(1/k)$ convergence rate. How critical? Consider, e.g., [105] in which the authors provide a simple example (with unbiased gradient estimates and $\mu = 1$) involving the minimization of a deterministic quadratic function *with only one optimization variable* in which c is overestimated, which results in β being underestimated. In the example, even after 10^9 iterations, the distance to the solution remains greater than 10^{-2}.

Role of the Initial Point Also observe the role played by the initial point, which determines the initial optimality gap, namely, $F(w_1)-F_*$. When using a fixed stepsize, the initial gap appears with an exponentially decreasing factor; see (4.14). In the case of diminishing stepsizes, the gap appears prominently in the second term defining ν in (4.22). However, with an appropriate initialization phase, one can easily diminish the role played by this term.[5] For example, suppose that one begins by running SG with a fixed stepsize $\bar{\alpha}$ until one (approximately) obtains a point, call it w_1, with $F(w_1) - F_* \leq \bar{\alpha}LM/(2c\mu)$. A guarantee for this bound can be argued from (4.14). Starting here with $\alpha_1 = \bar{\alpha}$, the choices for β and γ in Theorem 4.7 yield

$$(\gamma + 1)\mathbb{E}[F(w_1) - F_*] \leq \beta\alpha_1^{-1}\frac{\alpha_1 LM}{2c\mu} = \frac{\beta LM}{2c\mu} < \frac{\beta^2 LM}{2(\beta c\mu - 1)},$$

meaning that the value for ν is dominated by the first term in (4.22).

On a related note, we claim that for practical purposes the initial stepsize should be chosen as large as allowed, i.e., $\alpha_1 = \mu/(LM_G)$. Given this choice of α_1, the best asymptotic regime with decreasing stepsizes (4.21) is achieved by making ν as small as possible. Since we have argued that only the first term matters in the definition of ν, this leads to choosing $\beta = 2/(c\mu)$. Under these conditions, one has

$$\nu = \frac{\beta^2 LM}{2(\beta c\mu - 1)} = \frac{2}{\mu^2}\left(\frac{L}{c}\right)\left(\frac{M}{c}\right). \tag{4.24}$$

We shall see the (potentially large) ratios L/c and M/c arise again later.

Trade-Offs of (Mini-)Batching As a final observation about what can be learned from Theorems 4.6 and 4.7, let us take a moment to compare the theoretical performance of two fundamental algorithms—the simple SG iteration (3.7) and the mini-batch SG iteration (3.12)—when these results are applied for minimizing empirical risk, i.e., when $F = R_n$. This provides a glimpse into how such results can be used to compare algorithms in terms of their computational trade-offs.

The most elementary instance of our SG algorithm is simple SG, which, as we have seen, consists of picking a random sample index i_k at each iteration and computing

$$g(w_k, \xi_k) = \nabla f_{i_k}(w_k). \tag{4.25}$$

By contrast, instead of picking a single sample, mini-batch SG consists of randomly selecting a subset $\mathcal{S}_k$ of the sample indices and computing

$$g(w_k, \xi_k) = \frac{1}{|\mathcal{S}_k|}\sum_{i \in \mathcal{S}_k} \nabla f_i(w_k). \tag{4.26}$$

To compare these methods, let us assume for simplicity that we employ the same number of samples in each iteration so that the mini-batches are of constant size, i.e., $|S_k| = n_{\rm mb}$. There are then two distinct regimes to consider, namely, when $n_{\rm mb} \ll n$ and when $n_{\rm mb} \approx n$. Our goal here is to use the results of Theorems 4.6 and 4.7 to show that, in the former scenario, the theoretical benefit of mini-batching can appear to be somewhat ambiguous, meaning that one must leverage

[5] In fact, the bound (4.21) slightly overstates the asymptotic influence of the initial optimality gap. Applying Chung's lemma [36] to the contraction equation (4.23) shows that the first term in the definition of ν effectively determines the asymptotic convergence rate of $\mathbb{E}[F(w_k) - F_*]$.

certain computational tools to benefit from mini-batching in practice. As for the scenario when $n_{\rm mb} \approx n$, the comparison is more complex due to a trade-off between per-iteration costs and overall convergence rate of the method (recall §3.3). We leave a more formal treatment of this scenario, specifically with the goals of large-scale machine learning in mind, for §4.4.

Suppose then that the mini-batch size is $n_{\rm mb} \ll n$. The computation of the stochastic direction $g(w_k, \xi_k)$ in (4.26) is clearly $n_{\rm mb}$ times more expensive than in (4.25). In return, the variance of the direction is reduced by a factor of $1/n_{\rm mb}$. (See §5.2 for further discussion of this fact.) That is, with respect to our analysis, the constants M and M_V that appear in Assumption 4.3 (see (4.8)) are reduced by the same factor, becoming $M/n_{\rm mb}$ and $M_V/n_{\rm mb}$ for mini-batch SG. It is natural to ask whether this reduction of the variance pays for the higher per-iteration cost.

Consider, for instance, the case of employing a sufficiently small constant stepsize $\bar{\alpha} > 0$. For mini-batch SG, Theorem 4.6 leads to

$$\mathbb{E}[F(w_k) - F_*] \leq \frac{\bar{\alpha} L M}{2c\mu\, n_{\rm mb}} + [1 - \bar{\alpha} c\mu]^{k-1} \left(F(w_1) - F_* - \frac{\bar{\alpha} L M}{2c\mu\, n_{\rm mb}} \right).$$

Using the simple SG method with stepsize $\bar{\alpha}/n_{\rm mb}$ leads to a similar asymptotic gap:

$$\mathbb{E}[F(w_k) - F_*] \leq \frac{\bar{\alpha} L M}{2c\mu\, n_{\rm mb}} + \left[1 - \frac{\bar{\alpha} c\mu}{n_{\rm mb}}\right]^{k-1} \left(F(w_1) - F_* - \frac{\bar{\alpha} L M}{2c\mu\, n_{\rm mb}} \right).$$

The worse contraction constant (indicated using square brackets) means that one needs to run $n_{\rm mb}$ times more iterations of the simple SG algorithm to obtain an equivalent optimality gap. That said, since the computation in a simple SG iteration is $n_{\rm mb}$ times cheaper, this amounts to effectively the same total computation as for the mini-batch SG method. A similar analysis employing the result of Theorem 4.7 can be performed when decreasing stepsizes are used.

These observations suggest that the methods can be comparable. However, an important consideration remains. In particular, the convergence theorems require that the initial stepsize be smaller than $\mu/(LM_G)$. Since (4.9) shows that $M_G \geq \mu^2$, the largest this stepsize can be is $1/(\mu L)$. Therefore, one cannot simply assume that the mini-batch SG method is allowed to employ a stepsize that is $n_{\rm mb}$ times larger than the one used by SG. In other words, one cannot always compensate for the higher per-iteration cost of a mini-batch SG method by selecting a larger stepsize.

One can, however, realize benefits of mini-batching in practice since it offers important opportunities for software optimization and parallelization; e.g., using sizeable mini-batches is often the only way to fully leverage a GPU processor. Dynamic mini-batch sizes can also be used as a substitute for decreasing stepsizes; see §5.2.

4.3 SG for General Objectives

As mentioned in our case study of deep neural networks in §2.2, many important machine learning models lead to nonconvex optimization problems, which are currently having a profound impact in practice. Analyzing the SG method when minimizing nonconvex objectives is more challenging than in the convex case since such functions may possess multiple local minima and other stationary points. Still, we show in this subsection that one can provide meaningful guarantees for the SG method in nonconvex settings.

Paralleling §4.2, we present two results—one for employing a fixed positive stepsize and one for diminishing stepsizes. We maintain the same assumptions about the stochastic directions $g(w_k, \xi_k)$, but do not assume convexity of F. As before, the results in this section still apply to a wide class of methods since $g(w_k, \xi_k)$ could be defined as a (mini-batch) stochastic gradient or a Newton-like direction; recall (4.2).

Our first result describes the behavior of the sequence of gradients of F when fixed stepsizes are employed. Recall from Assumption 4.3 that the sequence of function values $\{F(w_k)\}$ is assumed to be bounded below by a scalar F_{inf}.

Theorem 4.8 (**Nonconvex Objective, Fixed Stepsize**). *Under Assumptions 4.1 and 4.3, suppose that the SG method (Algorithm 4.1) is run with a fixed stepsize, $\alpha_k = \bar{\alpha}$ for all $k \in \mathbb{N}$, satisfying*

$$0 < \bar{\alpha} \leq \frac{\mu}{LM_G}. \tag{4.27}$$

Then, the expected sum-of-squares and average-squared gradients of F corresponding to the SG iterates satisfy the following inequalities for all $K \in \mathbb{N}$:

$$\mathbb{E}\left[\sum_{k=1}^{K} \|\nabla F(w_k)\|_2^2\right] \leq \frac{K\bar{\alpha}LM}{\mu} + \frac{2(F(w_1) - F_{\text{inf}})}{\mu\bar{\alpha}} \tag{4.28a}$$

$$\text{and therefore } \mathbb{E}\left[\frac{1}{K}\sum_{k=1}^{K} \|\nabla F(w_k)\|_2^2\right] \leq \frac{\bar{\alpha}LM}{\mu} + \frac{2(F(w_1) - F_{\text{inf}})}{K\mu\bar{\alpha}} \tag{4.28b}$$

$$\xrightarrow{K\to\infty} \frac{\bar{\alpha}LM}{\mu}.$$

Proof. Taking the total expectation of (4.10b) and from (4.27),

$$\begin{aligned}\mathbb{E}[F(w_{k+1})] - \mathbb{E}[F(w_k)] &\leq -(\mu - \tfrac{1}{2}\bar{\alpha}LM_G)\bar{\alpha}\mathbb{E}[\|\nabla F(w_k)\|_2^2] + \tfrac{1}{2}\bar{\alpha}^2 LM \\ &\leq -\tfrac{1}{2}\mu\bar{\alpha}\mathbb{E}[\|\nabla F(w_k)\|_2^2] + \tfrac{1}{2}\bar{\alpha}^2 LM.\end{aligned}$$

Summing both sides of this inequality for $k \in \{1, \ldots, K\}$ and recalling Assumption 4.3(a) gives

$$F_{\text{inf}} - F(w_1) \leq \mathbb{E}[F(w_{K+1})] - F(w_1) \leq -\tfrac{1}{2}\mu\bar{\alpha}\sum_{k=1}^{K}\mathbb{E}[\|\nabla F(w_k)\|_2^2] + \tfrac{1}{2}K\bar{\alpha}^2 LM.$$

Rearranging yields (4.28a), and dividing further by K yields (4.28b). □

If $M = 0$, meaning that there is no noise or that noise reduces proportionally to $\|\nabla F(w_k)\|_2^2$ (see equations (4.8) and (4.9)), then (4.28a) captures a classical result for the full gradient method applied to nonconvex functions, namely, that the sum of squared gradients remains finite, implying that $\{\|\nabla F(w_k)\|_2\} \to 0$. In the presence of noise (i.e., $M > 0$), Theorem 4.8 illustrates the interplay between the stepsize $\bar{\alpha}$ and the variance of the stochastic directions. While one cannot bound the expected optimality gap as in the convex case, inequality (4.28b) bounds the average norm of the gradient of the objective function observed on $\{w_k\}$ visited during the first K iterations. This quantity gets smaller when K increases, indicating that the SG method spends increasingly more time in regions where the objective function has a (relatively) small gradient. Moreover, the asymptotic result that one obtains from (4.28b) illustrates that noise in the gradients inhibits further progress, as in (4.14) for the convex case. The average norm of the gradients can be made arbitrarily small by selecting a small stepsize, but doing so reduces the speed at which the norm of the gradient approaches its limiting distribution.

We now turn to the case when the SG method is applied to a nonconvex objective with a decreasing sequence of stepsizes satisfying the classical conditions (4.19). While not the strongest result that one can prove in this context—and, in fact, we prove a stronger result below—the following theorem is perhaps the easiest to interpret and remember. Hence, we state it first.

Theorem 4.9 (**Nonconvex Objective, Diminishing Stepsizes**). *Under Assumptions 4.1 and 4.3, suppose that the SG method (Algorithm 4.1) is run with a stepsize sequence satisfying* (4.19). *Then*

$$\liminf_{k\to\infty} \mathbb{E}[\|\nabla F(w_k)\|_2^2] = 0\,. \tag{4.29}$$

The proof of this theorem follows based on the results given in Theorem 4.10 below. A "lim inf" result of this type should be familiar to those knowledgeable of the nonlinear optimization literature. After all, such a result is all that can be shown for certain important methods, such as the nonlinear conjugate gradient method [114]. The intuition that one should gain from the statement of Theorem 4.9 is that, for the SG method with diminishing stepsizes, the expected gradient norms cannot stay bounded away from zero.

The following result characterizes more precisely the convergence property of SG.

Theorem 4.10 (**Nonconvex Objective, Diminishing Stepsizes**). *Under Assumptions 4.1 and 4.3, suppose that the SG method (Algorithm 4.1) is run with a stepsize sequence satisfying* (4.19). *Then, with* $A_K := \sum_{k=1}^K \alpha_k$*,*

$$\lim_{K\to\infty} \mathbb{E}\left[\sum_{k=1}^K \alpha_k \|\nabla F(w_k)\|_2^2\right] < \infty \tag{4.30a}$$

$$\textit{and therefore}\quad \mathbb{E}\left[\frac{1}{A_K}\sum_{k=1}^K \alpha_k \|\nabla F(w_k)\|_2^2\right] \xrightarrow{K\to\infty} 0. \tag{4.30b}$$

Proof. The second condition in (4.19) ensures that $\{\alpha_k\} \to 0$, meaning that, without loss of generality, we may assume that $\alpha_k L M_G \leq \mu$ for all $k \in \mathbb{N}$. Then, taking the total expectation of (4.10b),

$$\begin{aligned}\mathbb{E}[F(w_{k+1})] - \mathbb{E}[F(w_k)] &\leq -(\mu - \tfrac{1}{2}\alpha_k L M_G)\alpha_k\, \mathbb{E}[\|\nabla F(w_k)\|_2^2] + \tfrac{1}{2}\alpha_k^2 LM \\ &\leq -\tfrac{1}{2}\mu\alpha_k \mathbb{E}[\|\nabla F(w_k)\|_2^2] + \tfrac{1}{2}\alpha_k^2 LM\,.\end{aligned}$$

Summing both sides of this inequality for $k \in \{1, \dots, K\}$ gives

$$F_{\text{inf}} - \mathbb{E}[F(w_1)] \leq \mathbb{E}[F(w_{K+1})] - \mathbb{E}[F(w_1)] \leq -\tfrac{1}{2}\mu \sum_{k=1}^K \alpha_k \mathbb{E}[\|\nabla F(w_k)\|_2^2] + \tfrac{1}{2}LM\sum_{k=1}^K \alpha_k^2\,.$$

Dividing by $\mu/2$ and rearranging the terms, we obtain

$$\sum_{k=1}^K \alpha_k \mathbb{E}[\|\nabla F(w_k)\|_2^2] \;\leq\; \frac{2(\mathbb{E}[F(w_1)] - F_{\text{inf}})}{\mu} + \frac{LM}{\mu}\sum_{k=1}^K \alpha_k^2\,.$$

The second condition in (4.19) implies that the right-hand side of this inequality converges to a finite limit when K increases, proving (4.30a). Then, (4.30b) follows since the first condition in (4.19) ensures that $A_K \to \infty$ as $K \to \infty$. □

Theorem 4.10 establishes results about a weighted sum-of-squares and a weighted average of squared gradients of F similar to those in Theorem 4.8. However, unlike (4.28b), the conclusion (4.30b) states that the weighted average norm of the squared gradients converges to zero even if the gradients are noisy, i.e., if $M > 0$. The fact that (4.30b) only specifies a property of a weighted average (with weights dictated by $\{\alpha_k\}$) is only of minor importance since one can still conclude that the expected gradient norms cannot asymptotically stay far from zero.

We can now see that Theorem 4.9 is a direct consequence of Theorem 4.10, for if (4.29) did not hold, it would contradict Theorem 4.10.

The next result gives a stronger conclusion than Theorem 4.9, at the expense of only showing a property of the gradient of F at a randomly selected iterate.

Corollary 4.11. *Suppose the conditions of Theorem 4.10 hold. For any $K \in \mathbb{N}$, let $k(K) \in \{1, \dots, K\}$ represent a random index chosen with probabilities proportional to $\{\alpha_k\}_{k=1}^K$. Then, $\|\nabla F(w_{k(K)})\|_2 \xrightarrow{K\to\infty} 0$ in probability.*

Proof. Using Markov's inequality and (4.30a), for any $\varepsilon > 0$, we can write

$$\mathbb{P}\{\|\nabla F(w_k)\|_2 \geq \varepsilon\} = \mathbb{P}\{\|\nabla F(w_k)\|_2^2 \geq \varepsilon^2\} \leq \varepsilon^{-2}\,\mathbb{E}[\,\mathbb{E}_k[\|\nabla F(w_k)\|_2^2]\,] \xrightarrow{K\to\infty} 0\,,$$

which is the definition of convergence in probability. □

Finally, we present the following result (with proof in Appendix B) to illustrate that stronger convergence results also follow under additional regularity conditions.

Corollary 4.12. *Under the conditions of Theorem 4.10, if we further assume that the objective function F is twice differentiable, and that the mapping $w \mapsto \|\nabla F(w)\|_2^2$ has Lipschitz-continuous derivatives, then*

$$\lim_{k\to\infty} \mathbb{E}[\|\nabla F(w_k)\|_2^2] = 0.$$

4.4 Work Complexity for Large-Scale Learning

Our discussion thus far has focused on the convergence properties of SG when minimizing a given objective function representing either expected or empirical risk. However, our investigation would be incomplete without considering how these properties impact the computational workload associated with solving an underlying machine learning problem. As previewed in §3, there are arguments that a more slowly convergent algorithm such as SG, with its sublinear rate of convergence, is more efficient for large-scale learning than (full, batch) gradient-based methods that have a linear rate of convergence. The purpose of this section is to present these arguments in further detail.

As a first attempt for setting up a framework in which one may compare optimization algorithms for large-scale learning, one might be tempted to consider the situation of having a given training set size n and asking what type of algorithm—e.g., a simple SG or batch gradient method—would provide the best guarantees in terms of achieving a low expected risk. However, such a comparison is difficult to make when one cannot determine the precise trade-off between per-iteration costs and overall progress of the optimization method that one can guarantee.

An easier way to approach the issue is to consider a *big data* scenario with an infinite supply of training examples, but a limited computational time budget. One can then ask whether running a simple optimization algorithm such as SG works better than running a more sophisticated batch optimization algorithm. We follow such an approach now, following the work in [22, 23].

Suppose that both the expected risk R and the empirical risk R_n attain their minima with parameter vectors $w_* \in \arg\min R(w)$ and $w_n \in \arg\min R_n(w)$, respectively. In addition, let $\tilde{w}_n$ be the approximate empirical risk minimizer returned by a given optimization algorithm when the time budget $\mathcal{T}_{\max}$ is exhausted. The tradeoffs associated with this scenario can be formalized as choosing the family of prediction functions $\mathcal{H}$, the number of examples n, and the optimization accuracy $\epsilon := \mathbb{E}[R_n(\tilde{w}_n) - R_n(w_n)]$ in order to minimize, within time $\mathcal{T}_{max}$, the *total error*

$$\mathbb{E}[R(\tilde{w}_n)] = \underbrace{R(w_*)}_{\mathcal{E}_{app}(\mathcal{H})} + \underbrace{\mathbb{E}[R(w_n) - R(w_*)]}_{\mathcal{E}_{est}(\mathcal{H},n)} + \underbrace{\mathbb{E}[R(\tilde{w}_n) - R(w_n)]}_{\mathcal{E}_{opt}(\mathcal{H},n,\epsilon)}. \tag{4.31}$$

To minimize this error, one needs to balance the contributions from each of the three terms on the right-hand side. For instance, if one decides to make the optimization more accurate—i.e., reducing ϵ in the hope of also reducing the *optimization error* $\mathcal{E}_{opt}(\mathcal{H}, n, \epsilon)$ (evaluated with respect to R rather than R_n)—one might need to make up for the additional computing time by: (i) reducing the sample size n, potentially increasing the *estimation error* $\mathcal{E}_{est}(\mathcal{H}, n)$; or ($ii$) simplifying the function family $\mathcal{H}$, potentially increasing the *approximation error* $\mathcal{E}_{app}(\mathcal{H})$.

Useful guidelines for achieving an optimal balance between these errors can be obtained by setting aside the choice of $\mathcal{H}$ and carrying out a worst-case analysis on the influence of the sample size n and optimization tolerance ϵ, which together only influence the estimation and optimization errors. This simplified set-up can be formalized in terms of the macroscopic optimization problem

$$\min_{n,\epsilon} \ \mathcal{E}(n,\epsilon) = \mathbb{E}[R(\tilde{w}_n) - R(w_*)] \ \text{ s.t. } \ \mathcal{T}(n,\epsilon) \leq \mathcal{T}_{\max}. \tag{4.32}$$

The computing time $\mathcal{T}(n, \epsilon)$ depends on the details of the optimization algorithm in interesting ways. For example, the computing time of a batch algorithm increases linearly (at least) with the number of examples n, whereas, crucially, the computing time of a stochastic algorithm is independent of n. With a batch optimization algorithm, one could consider increasing ϵ in order to make time to use more training examples. However, with a stochastic algorithm, one should always use as many examples as possible because the per-iteration computing time is independent of n.

To be specific, let us compare the solutions of (4.32) for prototypical stochastic and batch methods—namely, simple SG and a batch gradient method—using simplified forms for the worst-case of the error function $\mathcal{E}$ and the time function $\mathcal{T}$. For the error function, a direct application of the uniform laws of large numbers [155] yields

$$\begin{aligned}
\mathcal{E}(n,\epsilon) = \mathbb{E}[R(\tilde{w}_n) - R(w_*)] \ &= \ \underbrace{\mathbb{E}[R(\tilde{w}_n) - R_n(\tilde{w}_n)]}_{= \mathcal{O}\left(\sqrt{\log(n)/n}\right)} + \underbrace{\mathbb{E}[R_n(\tilde{w}_n) - R_n(w_n)]}_{= \epsilon} \\
&+ \ \underbrace{\mathbb{E}[R_n(w_n) - R_n(w_*)]}_{\leq 0} + \underbrace{\mathbb{E}[R_n(w_*) - R(w_*)]}_{= \mathcal{O}\left(\sqrt{\log(n)/n}\right)},
\end{aligned}$$

which leads to the upper bound

$$\mathcal{E}(n,\epsilon) = \mathcal{O}\left(\sqrt{\frac{\log(n)}{n}} + \epsilon\right). \tag{4.33}$$

The inverse-square-root dependence on the number of examples n that appears here is typical of statistical problems. However, even faster convergence rates for reducing these terms with respect to n can be established under specific conditions. For instance, when the loss function is strongly convex [91] or when the data distribution satisfies certain assumptions [154], it is possible to show that

$$\mathcal{E}(n, \epsilon) = \mathcal{O}\left(\frac{\log(n)}{n} + \epsilon\right).$$

To simplify further, let us work with the asymptotic (i.e., for large n) equivalence

$$\mathcal{E}(n, \epsilon) \sim \frac{1}{n} + \epsilon, \tag{4.34}$$

which is the fastest rate that remains compatible with elementary statistical results.[6] Under this assumption, noting that the time constraint in (4.32) will always be active (since one can always lower ϵ, and hence $\mathcal{E}(n, \epsilon)$, by giving more time to the optimization algorithm), and recalling the worst-case computing time bounds introduced in §3.3, one arrives at the following conclusions.

- A simple SG method can achieve ϵ-optimality with a computing time of $\mathcal{T}_{\text{stoch}}(n, \epsilon) \sim 1/\epsilon$. Hence, within the time budget $\mathcal{T}_{\max}$, the accuracy achieved is proportional to $1/\mathcal{T}_{\max}$, regardless of the n. This means that, to minimize the error $\mathcal{E}(n, \epsilon)$, one should simply choose n as large as possible. Since the maximum number of examples that can be processed by SG during the time budget is proportional to $\mathcal{T}_{\max}$, it follows that the optimal error is proportional to $1/\mathcal{T}_{\max}$.

- A batch gradient method can achieve ϵ-optimality with a computing time of $\mathcal{T}_{\text{batch}}(n, \epsilon) \sim n \log(1/\epsilon)$. This means that, within the time budget $\mathcal{T}_{\max}$, it can achieve ϵ-optimality by processing $n \sim \mathcal{T}_{\max}/\log(1/\epsilon)$ examples. One now finds that the optimal error is not necessarily achieved by choosing n as large as possible, but rather by choosing ϵ (which dictates n) to minimize (4.34). Differentiating $\mathcal{E}(n, \epsilon) \sim \log(1/\epsilon)/\mathcal{T}_{\max} + \epsilon$ with respect to ϵ and setting the result equal to zero, one finds that optimality is achieved with $\epsilon \sim 1/\mathcal{T}_{\max}$, from which it follows that the optimal error is proportional to $\log(\mathcal{T}_{\max})/\mathcal{T}_{\max} + 1/\mathcal{T}_{\max}$.

These results are summarized in Table 4.1. Even though a batch approach possesses a better dependency on ϵ, this advantage does not make up for its dependence on n. This is true even though we have employed (4.34), the most favorable rate that one may reasonably assume. In conclusion, we have found that a stochastic optimization algorithm performs better in terms of expected error, and, hence, makes a better learning algorithm in the sense considered here. These observations are supported by practical experience (recall Figure 3.1 in §3.3).

A Lower Bound The results reported in Table 4.1 are also notable because the SG algorithm matches a lower complexity bound that has been established for any optimization method employing a *noisy oracle*. To be specific, in the widely employed model for studying optimization algorithms proposed by Nemirovsky and Yudin [106], one assumes that information regarding the objective function is acquired by querying an *oracle*, ignoring the computational demands of doing so. Using

[6] For example, suppose that one is estimating the mean of a distribution P defined on the real line by minimizing the risk $R(\mu) = \int (x - \mu)^2 dP(x)$. The convergence rate (4.34) amounts to estimating the distribution mean with a variance proportional to $1/n$. A faster convergence rate would violate the Cramer-Rao bound.

	Batch	Stochastic
$\mathcal{T}(n,\epsilon)$	$\sim n \log\left(\frac{1}{\epsilon}\right)$	$\frac{1}{\epsilon}$
$\mathcal{E}^*$	$\sim \frac{\log(\mathcal{T}_{\max})}{\mathcal{T}_{\max}} + \frac{1}{\mathcal{T}_{\max}}$	$\frac{1}{\mathcal{T}_{\max}}$

Table 4.1: The first row displays the computing times of idealized batch and stochastic optimization algorithms. The second row gives the corresponding solutions of (4.32), assuming (4.34).

such a model, it has been established, e.g., that the full gradient method applied to minimize a strongly convex objective function is not optimal in terms of the accuracy that can be achieved within a given number of calls to the oracle, but that one can achieve an optimal method through acceleration techniques; see §7.2.

The case when only gradient estimates are available through a noisy oracle has been studied, e.g. in [1, 128]. Roughly speaking, these investigations show that, again when minimizing a strongly convex function, no algorithm that performs k calls to the oracle can guarantee accuracy better than $\Omega(1/k)$. As we have seen, SG achieves this lower bound up to constant factors. This analysis applies both for the optimization of expected risk and empirical risk.

4.5 Commentary

Although the analysis presented in §4.4 can be quite compelling, it would be premature to conclude that SG is a perfect solution for large-scale machine learning problems. There is, in fact, a large gap between asymptotical behavior and practical realities. Next, we discuss issues related to this gap.

Fragility of the Asymptotic Performance of SG The convergence speed given, e.g., by Theorem 4.7, holds when the stepsize constant β exceeds a quantity inversely proportional to the strong convexity parameter c (see (4.20)). In some settings, determining such a value is relatively easy, such as when the objective function includes a squared ℓ_2-norm regularizer (e.g., as in (2.3)), in which case the regularization parameter provides a lower bound for c. However, despite the fact that this can work well in practice, it is not completely satisfactory because one should reduce the regularization parameter when the number of samples increases. It is therefore desirable to design algorithms that adapt to local convexity properties of the objective, so as to avoid having to place cumbersome restrictions on the stepsizes.

SG and Ill-Conditioning The analysis of §4.4 is compelling since, as long as the optimization problem is reasonably well-conditioned, the constant factors favor the SG algorithm. In particular, the minimalism of the SG algorithm allows for very efficient implementations that either fully leverage the sparsity of training examples (as in the case study on text classification in §2.1) or harness the computational power of GPU processors (as in the case study on deep neural network in §2.2). In contrast, state-of-the-art batch optimization algorithms often carry more overhead. However, this advantage erodes when the conditioning of the objective function worsens. Again, consider Theorem 4.7. This result involves constant factors that grow with both the condition number L/c and the ratio M/c. Both of these ratios can be improved greatly by adaptively rescaling the stochastic directions based on matrices that capture local curvature information of the objective function; see 6.

Opportunities for Distributed Computing Distributing the SG step computation can potentially reduce the computing time by a constant factor equal to the number of machines. However, such an improvement is difficult to realize. The SG algorithm is notoriously difficult to distribute efficiently because it accesses the shared parameter vector w with relatively high frequency. Consequently, even though it is very robust to additional noise and can be run with very relaxed synchronization [112, 45], distributed SG algorithms suffer from large communication overhead. Since this overhead is potentially much larger than the additional work associated with mini-batch and other methods with higher per-iteration costs, distributed computing offers new opportunities for the success of such methods for machine learning.

Alternatives with Faster Convergence As mentioned above, [1, 128] establish lower complexity bounds for optimization algorithms that only access information about the objective function through noisy estimates of $F(w_k)$ and $\nabla F(w_k)$ acquired in each iteration. The bounds apply, e.g., when SG is employed to minimize the expected risk R using gradient estimates evaluated on samples drawn from the distribution P. However, an algorithm that optimizes the empirical risk R_n has access to an additional piece of information: it knows when a gradient estimate is evaluated on a training example that has already been visited during previous iterations. Recent *gradient aggregation* methods (see §5.3) make use of this information and improve upon the lower bound in [1] for the optimization of the empirical risk (though not for the expected risk). These algorithms enjoy linear convergence with low computing times in practice. Another avenue for improving the convergence rate is to employ a *dynamic sampling* approach (see §5.2), which, as we shall see, can match the optimal asymptotic efficiency of SG in big data settings.

Inset 4.3: Regret Bounds

Convergence results for SG and its variants are occasionally established using *regret bounds* as an intermediate step [164, 144, 54]. Regret bounds can be traced to Novikoff's analysis of the Perceptron [115] and to Cover's universal portfolios [41]. To illustrate, suppose that one aims to minimize a convex expected risk measure $R(w) = \mathbb{E}[f(w;\xi)]$ over $w \in \mathbb{R}^d$ with minimizer $w_* \in \mathbb{R}^d$. At a given iterate w_k, one obtains by convexity of $f(w;\xi_k)$ (recall (A.1)) that

$$\|w_{k+1} - w_*\|^2 - \|w_k - w_*\|^2 \leq -2\alpha_k(f(w_k;\xi_k) - f(w_*;\xi_k)) + \alpha_k^2\|\nabla f(w_k;\xi_k)\|_2^2.$$

Following [164], assuming that $\|\nabla f(w_k;\xi_k)\|_2^2 \leq M$ and $\|w_k - w_*\|_2^2 < B$ for some constants $M > 0$ and $B > 0$ for all $k \in \mathbb{N}$, one finds that

$$\begin{aligned}
&\alpha_{k+1}^{-1}\|w_{k+1}-w_*\|_2^2 - \alpha_k^{-1}\|w_k-w_*\|_2^2 \\
\leq\ & -2(f(w_k;\xi_k) - f(w_*;\xi_k)) + \alpha_k M + (\alpha_{k+1}^{-1} - \alpha_k^{-1})\|w_k - w_*\|_2^2 \\
\leq\ & -2(f(w_k;\xi_k) - f(w_*;\xi_k)) + \alpha_k M + (\alpha_{k+1}^{-1} - \alpha_k^{-1})B.
\end{aligned}$$

Summing for $k = \{1 \dots K\}$ with stepsizes $\alpha_k = 1/\sqrt{k}$ leads to the regret bound

$$\left(\sum_{k=1}^{K} f(w_k;\xi_k)\right) \leq \left(\sum_{k=1}^{K} f(w_*;\xi_k)\right) + M\sqrt{K} + o(\sqrt{K}), \tag{4.35}$$

which bounds the losses incurred from $\{w_k\}$ compared to those yielded by the fixed vector w_*. Such a bound is remarkable because its derivation holds for any sequence of noise variables $\{\xi_k\}$. This means that the average loss observed during the execution of the SG algorithm is *never* much worse than the best average loss one would have obtained if the optimal parameter w_* were known in advance. Further assuming that the noise variables are independent and using a martingale argument [34] leads to more traditional results of the form

$$\mathbb{E}\left[\frac{1}{K}\sum_{k=1}^{K} F(w_k)\right] \leq F_* + \mathcal{O}\left(\frac{1}{\sqrt{K}}\right).$$

As long as one makes the same independent noise assumption, results obtained with this technique cannot be fundamentally different from the results that we have established. However, one should appreciate that the regret bound (4.35) itself remains meaningful when the noise variables are dependent or even adversarial. Such results reveals interesting connections between probability theory, machine learning, and game theory [143, 34].

5 Noise Reduction Methods

The theoretical arguments in the previous section, together with extensive computational experience, have led many in the machine learning community to view SG as the ideal optimization approach for large-scale applications. We argue, however, that this is far from settled. SG suffers from, amongst other things, the adverse effect of noisy gradient estimates. This prevents it from converging to the solution when fixed stepsizes are used and leads to a slow, sublinear rate of convergence when a diminishing stepsize sequence $\{\alpha_k\}$ is employed.

To address this limitation, methods endowed with *noise reduction* capabilities have been developed. These methods, which reduce the errors in the gradient estimates and/or iterate sequence, have proved to be effective in practice and enjoy attractive theoretical properties. Recalling the schematic of optimization methods in Figure 3.3, we depict these methods on the horizontal axis given in Figure 5.1.

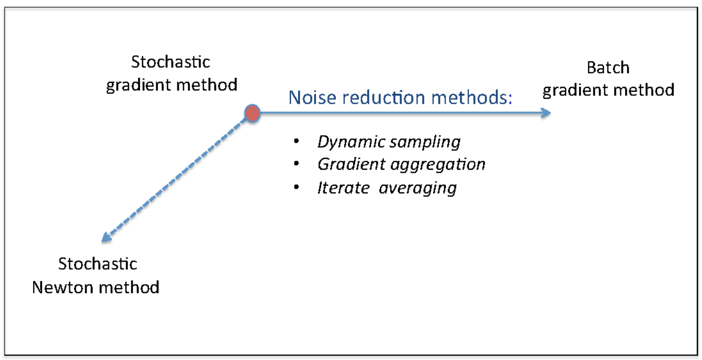

Fig. 5.1: View of the schematic from Figure 3.3 with a focus on noise reduction methods.

The first two classes of methods that we consider achieve noise reduction in a manner that allows them to possess a linear rate of convergence to the optimal value using a fixed stepsize. The first type, *dynamic sampling* methods, achieve noise reduction by gradually increasing the mini-batch size used in the gradient computation, thus employing increasingly more accurate gradient estimates as the optimization process proceeds. *Gradient aggregation* methods, on the other hand, improve the quality of the search directions by storing gradient estimates corresponding to samples employed in previous iterations, updating one (or some) of these estimates in each iteration, and defining the search direction as a weighted average of these estimates.

The third class of methods that we consider, *iterate averaging* methods, accomplish noise reduction not by averaging gradient estimates, but by maintaining an average of iterates computed during the optimization process. Employing a more aggressive stepsize sequence—of order $\mathcal{O}(1/\sqrt{k})$ rather than $\mathcal{O}(1/k)$, which is appealing in itself—it is this sequence of averaged iterates that converges to the solution. These methods are closer in spirit to SG and their rate of convergence remains sublinear, but it can be shown that the variance of the sequence of average iterates is smaller than the variance of the SG iterates.

To formally motivate a concept of noise reduction, we begin this section by discussing a fundamental result that stipulates a rate of decrease in noise that allows a stochastic-gradient-type method to converge at a linear rate. We then show that a dynamic sampling method that enforces such noise reduction enjoys optimal complexity bounds, as defined in §4.4. Next, we discuss three gradient aggregation methods—SVRG, SAGA, and SAG—the first of which can be viewed as a bridge between methods that control errors in the gradient with methods like SAGA and SAG in which noise reduction is accomplished in a more subtle manner. We conclude with a discussion of the practical and theoretical properties of iterate averaging methods.

5.1 Reducing Noise at a Geometric Rate

Let us recall the fundamental inequality (4.4), which we restate here for convenience:

$$\mathbb{E}_{\xi_k}[F(w_{k+1})] - F(w_k) \le -\alpha_k \nabla F(w_k)^T \mathbb{E}_{\xi_k}[g(w_k,\xi_k)] + \tfrac{1}{2}\alpha_k^2 L \mathbb{E}_{\xi_k}[\|g(w_k,\xi_k)\|_2^2].$$

(Recall that, as stated in (4.1), the objective F could stand for the expected risk R or empirical risk R_n.) It is intuitively clear that if $-g(w_k,\xi_k)$ is a descent direction in expectation (making the first term on the right hand side negative) and if we are able to decrease $\mathbb{E}_{\xi_k}[\|g(w_k,\xi_k)\|_2^2$ fast enough (along with $\|\nabla F(w_k)\|_2^2$), then the effect of having noisy directions will not impede a fast rate of convergence. From another point of view, we can expect such behavior if, in Assumption 4.3, we suppose instead that the variance of $g(w_k,\xi_k)$ vanishes sufficiently quickly.

We formalize this intuitive argument by considering the SG method with a fixed stepsize and showing that the sequence of expected optimality gaps vanishes at a linear rate as long as the variance of the stochastic vectors, denoted by $\mathbb{V}_{\xi_k}[g(w_k,\xi_k)]$ (recall (4.6)), decreases *geometrically.*

Theorem 5.1 (Strongly Convex Objective, Noise Reduction). *Suppose that Assumptions 4.1, 4.3, and 4.5 (with $F_{\rm inf} = F_*$) hold, but with (4.8) refined to the existence of constants $M \ge 0$ and $\zeta \in (0,1)$ such that, for all $k \in \mathbb{N}$,*

$$\mathbb{V}_{\xi_k}[g(w_k,\xi_k)] \le M\zeta^{k-1}. \tag{5.1}$$

In addition, suppose that the SG method (Algorithm 4.1) is run with a fixed stepsize, $\alpha_k = \bar\alpha$ for all $k \in \mathbb{N}$, satisfying

$$0 < \bar\alpha \le \min\left\{\frac{\mu}{L\mu_G^2}, \frac{1}{c\mu}\right\}. \tag{5.2}$$

Then, for all $k \in \mathbb{N}$, the expected optimality gap satisfies

$$\mathbb{E}[F(w_k) - F_*] \le \omega\rho^{k-1}, \tag{5.3}$$

where

$$\omega := \max\{\tfrac{\bar\alpha L M}{c\mu}, F(w_1) - F_*\} \tag{5.4a}$$

$$\textit{and}\ \ \rho := \max\{1 - \tfrac{\bar\alpha c\mu}{2}, \zeta\} < 1. \tag{5.4b}$$

Proof. By Lemma 4.4 (specifically, (4.10a)), we have

$$\mathbb{E}_{\xi_k}[F(w_{k+1})] - F(w_k) \le -\mu\bar\alpha\|\nabla F(w_k)\|_2^2 + \tfrac{1}{2}\bar\alpha^2 L \mathbb{E}_{\xi_k}[\|g(w_k,\xi_k)\|_2^2].$$

Hence, from (4.6), (4.7b), (5.1), (5.2), and (4.12), we have

$$\begin{aligned}
\mathbb{E}_{\xi_k}[F(w_{k+1})] - F(w_k) &\leq -\mu\bar{\alpha}\|\nabla F(w_k)\|_2^2 + \tfrac{1}{2}\bar{\alpha}^2 L \left(\mu_G^2\|\nabla F(w_k)\|_2^2 + M\zeta^{k-1}\right) \\
&\leq -(\mu - \tfrac{1}{2}\bar{\alpha}L\mu_G^2)\bar{\alpha}\|\nabla F(w_k)\|_2^2 + \tfrac{1}{2}\bar{\alpha}^2 LM\zeta^{k-1} \\
&\leq -\tfrac{1}{2}\mu\bar{\alpha}\|\nabla F(w_k)\|_2^2 + \tfrac{1}{2}\bar{\alpha}^2 LM\zeta^{k-1} \\
&\leq -\bar{\alpha}c\mu(F(w_k) - F_*) + \tfrac{1}{2}\bar{\alpha}^2 LM\zeta^{k-1}.
\end{aligned}$$

Adding and subtracting F_* and taking total expectations, this yields

$$\mathbb{E}[F(w_{k+1}) - F_*] \leq (1 - \bar{\alpha}c\mu)\mathbb{E}[F(w_k) - F_*] + \tfrac{1}{2}\bar{\alpha}^2 LM\zeta^{k-1}. \tag{5.5}$$

We now use induction to prove (5.3). By the definition of ω, it holds for $k = 1$. Then, assuming it holds for $k \geq 1$, we have from (5.5), (5.4a), and (5.4b) that

$$\begin{aligned}
\mathbb{E}[F(w_{k+1}) - F_*] &\leq (1 - \bar{\alpha}c\mu)\,\omega\rho^{k-1} + \tfrac{1}{2}\bar{\alpha}^2 LM\zeta^{k-1} \\
&= \omega\rho^{k-1}\left(1 - \bar{\alpha}c\mu + \frac{\bar{\alpha}^2 LM}{2\omega}\left(\frac{\zeta}{\rho}\right)^{k-1}\right) \\
&\leq \omega\rho^{k-1}\left(1 - \bar{\alpha}c\mu + \frac{\bar{\alpha}^2 LM}{2\omega}\right) \\
&\leq \omega\rho^{k-1}\left(1 - \bar{\alpha}c\mu + \frac{\bar{\alpha}c\mu}{2}\right) \\
&= \omega\rho^{k-1}\left(1 - \frac{\bar{\alpha}c\mu}{2}\right) \\
&\leq \omega\rho^k,
\end{aligned}$$

as desired. □

Consideration of the typical magnitudes of the constants μ, L, μ_G, and c in (5.2) reveals that the admissible range of values of $\bar{\alpha}$ is large, i.e., the restriction on the stepsize $\bar{\alpha}$ is not unrealistic in practical situations.

Now, a natural question is whether one can design efficient optimization methods for attaining the critical bound (5.1) on the variance of the stochastic directions. We show next that a dynamic sampling method is one such approach.

5.2 Dynamic Sample Size Methods

Consider the iteration

$$w_{k+1} \leftarrow w_k - \bar{\alpha}g(w_k, \xi_k), \tag{5.6}$$

where the stochastic directions are computed for some $\tau > 1$ as

$$g(w_k, \xi_k) := \frac{1}{n_k}\sum_{i\in\mathcal{S}_k}\nabla f(w_k; \xi_{k,i}) \quad \text{with} \quad n_k := |\mathcal{S}_k| = \lceil\tau^{k-1}\rceil. \tag{5.7}$$

That is, consider a mini-batch SG iteration with a fixed stepsize in which the mini-batch size used to compute unbiased stochastic gradient estimates increases geometrically as a function of the iteration counter k. To show that such an approach can fall under the class of methods considered in Theorem 5.1, suppose that the samples represented by the random variables $\{\xi_{k,i}\}_{i\in\mathcal{S}_k}$ are drawn independently according to P for all $k \in \mathbb{N}$. If we assume that each stochastic gradient $\nabla f(w_k;\xi_{k,i})$ has an expectation equal to the true gradient $\nabla F(w_k)$, then (4.7a) holds with $\mu_G = \mu = 1$. If, in addition, the variance of each such stochastic gradient is equal and is bounded by $M \geq 0$, then for arbitrary $i \in \mathcal{S}_k$ we have (see [61, p.183])

$$\mathbb{V}_{\xi_k}[g(w_k,\xi_k)] \leq \frac{\mathbb{V}_{\xi_k}[\nabla f(w_k;\xi_{k,i})]}{n_k} \leq \frac{M}{n_k}. \tag{5.8}$$

This bound, when combined with the rate of increase in n_k given in (5.7), yields (5.1). We have thus shown that if one employs a mini-batch SG method with (unbiased) gradient estimates computed as in (5.7), then, by Theorem 5.1, one obtains linear convergence to the optimal value of a strongly convex function. We state this formally as the following corollary to Theorem 5.1.

Corollary 5.2. *Let $\{w_k\}$ be the iterates generated by (5.6)–(5.7) with unbiased gradient estimates, i.e., $\mathbb{E}_{\xi_{k,i}}[\nabla f(w_k;\xi_{k,i})] = \nabla F(w_k)$ for all $k \in \mathbb{N}$ and $i \in \mathcal{S}_k$. Then, the variance condition (5.1) is satisfied, and if all other assumptions of Theorem 5.1 hold, then the expected optimality gap vanishes linearly in the sense of (5.3).*

The reader may question whether it is meaningful to describe a method as linearly convergent if the per-iteration cost increases without bound. In other words, it is not immediately apparent that such an algorithm is competitive with a classical SG approach even though the desired reduction in the gradient variance is achieved. To address this question, let us estimate the total work complexity of the dynamic sampling algorithm, defined as the number of evaluations of the individual gradients $\nabla f(w_k;\xi_{k,i})$ required to compute an ϵ-optimal solution, i.e., to achieve

$$\mathbb{E}[F(w_k) - F_*] \leq \epsilon. \tag{5.9}$$

We have seen that the simple SG method (3.7) requires one such evaluation per iteration, and that its rate of convergence for diminishing stepsizes (i.e., the only set-up in which convergence to the solution can be guaranteed) is given by (4.21). Therefore, as previously mentioned, the number of stochastic gradient evaluations required by the SG method to guarantee (5.9) is $\mathcal{O}(\epsilon^{-1})$. We now show that the method (5.6)–(5.7) can attain the same complexity.

Theorem 5.3. *Suppose that the dynamic sampling SG method (5.6)–(5.7) is run with a stepsize $\bar{\alpha}$ satisfying (5.2) and some $\tau \in (1, (1-\frac{\bar{\alpha}c\mu}{2})^{-1}]$. In addition, suppose that Assumptions 4.1, 4.3, and Assumption 4.5 (with $F_{\rm inf} = F_*$) hold. Then, the total number of evaluations of a stochastic gradient of the form $\nabla f(w_k;\xi_{k,i})$ required to obtain (5.9) is $\mathcal{O}(\epsilon^{-1})$.*

Proof. We have that the conditions of Theorem 5.1 hold with $\zeta = 1/\tau$. Hence, we have from (5.3) that there exists $\bar{k} \in \mathbb{N}$ such that (5.9) holds for all $k \geq \bar{k}$. We can then use $\omega\rho^{\bar{k}-1} \leq \epsilon$ to write $(\bar{k}-1)\log\rho \leq \log(\epsilon/\omega)$, which along with $\rho \in (0,1)$ (recall (5.4b)) implies that

$$\bar{k} - 1 \geq \left\lceil \frac{\log(\epsilon/\omega)}{\log\rho} \right\rceil = \left\lceil \frac{\log(\omega/\epsilon)}{-\log\rho} \right\rceil. \tag{5.10}$$

Let us now estimate the total number of sample gradient evaluations required up through iteration $\bar{k}$. We claim that, without loss of generality, we may assume that $\log(\omega/\epsilon)/(-\log\rho)$ is integer-valued and that (5.10) holds at equality. Then, by (5.7), the number of sample gradients required in iteration $\bar{k}$ is $\lceil \tau^{\bar{k}-1} \rceil$ where

$$\begin{aligned}
\tau^{\bar{k}-1} &= \tau^{\frac{\log(\omega/\epsilon)}{-\log\rho}} \\
&= \exp\left(\log\left(\tau^{\frac{\log(\omega/\epsilon)}{-\log\rho}}\right)\right) \\
&= \exp\left(\left(\frac{\log(\omega/\epsilon)}{-\log\rho}\right)\log\tau\right) \\
&= (\exp(\log(\omega/\epsilon)))^{\frac{\log\tau}{-\log\rho}} \\
&= \left(\frac{\omega}{\epsilon}\right)^{\theta} \quad \text{with} \quad \theta := \frac{\log\tau}{-\log\rho}.
\end{aligned}$$

Therefore, the total number of sample gradient evaluations for the first $\bar{k}$ iterations is

$$\sum_{j=1}^{\bar{k}} \lceil \tau^{j-1} \rceil \leq 2\sum_{j=1}^{\bar{k}} \tau^{j-1} = 2\left(\frac{\tau^{\bar{k}} - \tau}{\tau - 1}\right) = 2\left(\frac{\tau(\omega/\epsilon)^{\theta} - \tau}{\tau - 1}\right) \leq 2\left(\frac{\omega}{\epsilon}\right)^{\theta}\left(\frac{1}{1 - 1/\tau}\right).$$

In fact, since $\tau \leq (1 - \frac{\bar{\alpha}c\mu}{2})^{-1}$, it follows from (5.4b) that $\rho = \zeta = \tau^{-1}$, which implies that $\theta = 1$. Specifically, with $\tau = (1 - \sigma(\frac{\bar{\alpha}c\mu}{2}))^{-1}$ for some $\sigma \in [0, 1]$, then $\theta = 1$ and

$$\sum_{j=1}^{\bar{k}} \lceil \tau^{j-1} \rceil \leq \frac{4\omega}{\sigma\epsilon\bar{\alpha}c\mu},$$

as desired. □

The discussion so far has focused on dynamic sampling strategies for a gradient method, but these ideas also apply for second-order methods that incorporate the matrix H as in (4.2).

This leads to the following question: given the rate of convergence of a batch optimization algorithm on strongly convex functions (i.e., linear, superlinear, etc.), what should be the sampling rate so that the overall algorithm is *efficient* in the sense that it results in the lowest computational complexity? To answer this question, certain results have been established [120]: *(i)* if the optimization method has a sublinear rate of convergence, then there is no sampling rate that makes the algorithm "efficient"; *(ii)* if the optimization algorithm is linearly convergent, then the sampling rate must be geometric (with restrictions on the constant in the rate) for the algorithm to be "efficient"; *(iii)* for superlinearly convergent methods, increasing the sample size at a rate that is slightly faster than geometric will yield an "efficient" method.

5.2.1 Practical Implementation

We now address the question of how to design practical algorithms that enjoy the theoretical benefits of noise reduction through the use of increasing sample sizes.

One approach is to follow the theoretical results described above and tie the rate of growth in the sample size $n_k = |\mathcal{S}_k|$ to the rate of convergence of the optimization iteration [62, 77]. Such an approach, which *presets* the sampling rate before running the optimization algorithm, requires some experimentation. For example, for the iteration (5.6), one needs to find a value of the parameter $\tau > 1$ in (5.7) that yields good performance for the application at hand. In addition, one may want to delay the application of dynamic sampling to prevent the full sample set from being employed too soon (or at all). Such heuristic adaptations could be difficult in practice.

An alternative is to consider mechanisms for choosing the sample sizes not according to a prescribed sequence, but *adaptively* according to information gathered during the optimization process. One avenue that has been explored along these lines has been to design techniques that produce descent directions sufficiently often [28, 73]. Such an idea is based on two observations. First, one can show that *any* direction $g(w_k, \xi_k)$ is a descent direction for F at w_k if, for some $\chi \in [0, 1)$, one has

$$\delta(w_k, \xi_k) := \|g(w_k, \xi_k) - \nabla F(w_k)\|_2 \leq \chi \|g(w_k, \xi_k)\|_2. \tag{5.11}$$

To see this, note that $\|\nabla F(w_k)\|_2 \geq (1-\chi)\|g(w_k, \xi_k)\|_2$, which after squaring (5.11) implies

$$\begin{aligned}\nabla F(w_k)^T g(w_k, \xi_k) &\geq \tfrac{1}{2}(1-\chi^2)\|g(w_k, \xi_k)\|_2^2 + \|\nabla F(w_k)\|_2^2 \\ &\geq \tfrac{1}{2}(1-\chi^2 + (1-\chi)^2)\|g(w_k, \xi_k)\|_2^2 \\ &\geq (1-\chi)\|g(w_k, \xi_k)\|_2^2.\end{aligned}$$

The second observation is that while one cannot cheaply verify the inequality in (5.11) because it involves the evaluation of $\nabla F(w_k)$, one can estimate the left-hand side $\delta(w_k, \xi_k)$, and then choose n_k so that (5.11) holds sufficiently often.

Specifically, if we assume that $g(w_k, \xi_k)$ is an unbiased gradient estimate, then

$$\mathbb{E}[\delta(w_k, \xi_k)^2] = \mathbb{E}[\|g(w_k, \xi_k) - \nabla F(w_k)\|_2^2] = \mathbb{V}_{\xi_k}[g(w_k, \xi_k)].$$

This variance is expensive to compute, but one can approximate it with a sample variance. For example, if the samples are drawn without replacement from a set of (very large) size n_k, then one has the approximation

$$\mathbb{V}_{\xi_k}[g(w_k, \xi_k)] \approx \frac{\text{trace}(\text{Cov}(\{\nabla f(w_k; \xi_{k,i})\}_{i \in \mathcal{S}_k}))}{n_k} =: \varphi_k.$$

An *adaptive sampling* algorithm thus tests the following condition in place of (5.11):

$$\varphi_k \leq \chi^2 \|g(w_k, \xi_k)\|_2^2. \tag{5.12}$$

If this condition is not satisfied, then one may consider increasing the sample size—either immediately in iteration k or in a subsequent iteration—to a size that one might predict would satisfy such a condition. This technique is algorithmically attractive, but does not guarantee that the sample size n_k increases at a geometric rate. One can, however, employ a backup [28, 73]: if (5.12) increases the sampling rate more slowly than a preset geometric sequence, then a growth in the sample size is imposed.

Dynamic sampling techniques are not yet widely used in machine learning, and we suspect that the practical technique presented here might merely serve as a starting point for further investigations. Ultimately, an algorithm that performs like SG at the start and transitions to a regime of reduced variance in an efficient manner could prove to be a very powerful method for large-scale learning.

5.3 Gradient Aggregation

The dynamic sample size methods described in the previous subsection require a careful balance in order to achieve the desired linear rate of convergence without jeopardizing per-iteration costs. Alternatively, one can attempt to achieve an improved rate by asking a different question: rather than compute increasingly more *new* stochastic gradient information in each iteration, is it possible to achieve a lower variance by *reusing* and/or *revising* previously computed information? After all, if the current iterate has not been displaced too far from previous iterates, then stochastic gradient information from previous iterates may still be useful. In addition, if one maintains indexed gradient estimates in storage, then one can revise specific estimates as new information is collected. Ideas such as these lead to concepts of *gradient aggregation*. In this subsection, we present ideas for gradient aggregation in the context of minimizing a finite sum such as an empirical risk measure R_n, for which they were originally designed.

Gradient aggregation algorithms for minimizing finite sums that possess cheap per-iteration costs have a long history. For example, Bertsekas and co-authors [13] have proposed *incremental gradient* methods, the randomized versions of which can be viewed as instances of a basic SG method for minimizing a finite sum. However, the basic variants of these methods only achieve a sublinear rate of convergence. By contrast, the methods on which we focus in this section are able to achieve a linear rate of convergence on strongly convex problems. This improved rate is achieved primarily by either an increase in computation or an increase in storage.

5.3.1 SVRG

The first method we consider operates in cycles. At the beginning of each cycle, an iterate w_k is available at which the algorithm computes a batch gradient $\nabla R_n(w_k) = \frac{1}{n}\sum_{i=1}^n \nabla f_i(w_k)$. Then, after initializing $\tilde{w}_1 \leftarrow w_k$, a set of m inner iterations indexed by j with an update $\tilde{w}_{j+1} \leftarrow \tilde{w}_j - \alpha \tilde{g}_j$ are performed, where

$$\tilde{g}_j \leftarrow \nabla f_{i_j}(\tilde{w}_j) - (\nabla f_{i_j}(w_k) - \nabla R_n(w_k)) \tag{5.13}$$

and $i_j \in \{1, \dots, n\}$ is chosen at random. This formula has a simple interpretation. Since the expected value of $\nabla f_{i_j}(w_k)$ over all possible $i_j \in \{1, \dots, n\}$ is equal to $\nabla R_n(w_k)$, one can view $\nabla f_{i_j}(w_k) - \nabla R_n(w_k)$ as the bias in the gradient estimate $\nabla f_{i_j}(w_k)$. Thus, in every iteration, the algorithm randomly draws a stochastic gradient $\nabla f_{i_j}(\tilde{w}_j)$ evaluated at the current inner iterate $\tilde{w}_j$ and corrects it based on a perceived bias. Overall, $\tilde{g}_j$ represents an unbiased estimator of $\nabla R_n(\tilde{w}_j)$, but with a variance that one can expect to be smaller than if one were simply to chose $\tilde{g}_j = \nabla f_{i_j}(\tilde{w}_j)$ (as in simple SG). This is the reason that the method is referred to as the *stochastic variance reduced gradient* (SVRG) method.

A formal description of a few variants of SVRG is presented as Algorithm 5.1. For both options (b) and (c), it has been shown that when applied to minimize a strongly convex R_n, Algorithm 5.1 can achieve a linear rate of convergence [82]. More precisely, if the stepsize α and the length of the inner cycle m are chosen so that

$$\rho := \frac{1}{1 - 2\alpha L}\left(\frac{1}{mc\alpha} + 2L\alpha\right) < 1,$$

then, given that the algorithm has reached w_k, one obtains

$$\mathbb{E}[R_n(w_{k+1}) - R_n(w_*)] \le \rho \mathbb{E}[R_n(w_k) - R_n(w_*)]$$

(where expectation is taken with respect to the random variables in the inner cycle). It should be emphasized that resulting linear convergence rate applies to the outer iterates $\{w_k\}$, where each step from w_k to w_{k+1} requires $2m+n$ evaluations of component gradients: Step 7 requires two stochastic gradient evaluations while Step 3 requires n (a full gradient). Therefore, one iteration of SVRG is much more expensive than one in SG, and in fact is comparable to a full gradient iteration.

Algorithm 5.1 SVRG Methods for Minimizing an Empirical Risk R_n

1: Choose an initial iterate $w_1 \in \mathbb{R}^d$, stepsize $\alpha > 0$, and positive integer m.
2: **for** $k = 1, 2, \dots$ **do**
3: Compute the batch gradient $\nabla R_n(w_k)$.
4: Initialize $\tilde{w}_1 \leftarrow w_k$.
5: **for** $j = 1, \dots, m$ **do**
6: Chose i_j uniformly from $\{1, \dots, n\}$.
7: Set $\tilde{g}_j \leftarrow \nabla f_{i_j}(\tilde{w}_j) - (\nabla f_{i_j}(w_k) - \nabla R_n(w_k))$.
8: Set $\tilde{w}_{j+1} \leftarrow \tilde{w}_j - \alpha \tilde{g}_j$.
9: **end for**
10: Option (a): Set $w_{k+1} = \tilde{w}_{m+1}$
11: Option (b): Set $w_{k+1} = \frac{1}{m}\sum_{j=1}^{m} \tilde{w}_{j+1}$
12: Option (c): Choose j uniformly from $\{1, \dots, m\}$ and set $w_{k+1} = \tilde{w}_{j+1}$.
13: **end for**

In practice, SVRG appears to be quite effective in certain applications compared with SG if one requires high training accuracy. For the first epochs, SG is more efficient, but once the iterates approach the solution the benefits of the fast convergence rate of SVRG can be observed. Without explicit knowledge of L and c, the length of the inner cycle m and the stepsize α are typically both chosen by experimentation.

5.3.2 SAGA

The second method we consider employs an iteration that is closer in form to SG in that it does not operate in cycles, nor does it compute batch gradients (except possibly at the initial point). Instead, in each iteration, it computes a stochastic vector g_k as the average of stochastic gradients evaluated at previous iterates. Specifically, in iteration k, the method will have stored $\nabla f_i(w_{[i]})$ for all $i \in \{1, \dots, n\}$ where $w_{[i]}$ represents the latest iterate at which ∇f_i was evaluated. An integer $j \in \{1, \dots, n\}$ is then chosen at random and the stochastic vector is set by

$$g_k \leftarrow \nabla f_j(w_k) - \nabla f_j(w_{[j]}) + \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(w_{[i]}). \tag{5.14}$$

Taking the expectation of g_k with respect to all choices of $j \in \{1, \dots, n\}$, one again has that $\mathbb{E}[g_k] = \nabla R_n(w_k)$. Thus, the method employs unbiased gradient estimates, but with variances that are expected to be less than the stochastic gradients that would be employed in a basic SG routine. A precise algorithm employing (5.14), referred to as SAGA [46], is given in Algorithm 5.2.

Beyond its initialization phase, the per-iteration cost of Algorithm 5.2 is the same as in a basic SG method. However, it has been shown that the method can achieve a linear rate of convergence

Algorithm 5.2 SAGA Method for Minimizing an Empirical Risk R_n

```
Choose an initial iterate w_1 ∈ R^d and stepsize α > 0.
for i = 1, ..., n do
    Compute ∇f_i(w_1).
    Store ∇f_i(w_[i]) ← ∇f_i(w_1).
end for
for k = 1, 2, ... do
    Choose j uniformly in {1, ..., n}.
    Compute ∇f_j(w_k).
    Set g_k ← ∇f_j(w_k) − ∇f_j(w_[j]) + (1/n) Σ_{i=1}^n ∇f_i(w_[i]).
    Store ∇f_j(w_[j]) ← ∇f_j(w_k).
    Set w_{k+1} ← w_k − α g_k.
end for
```

when minimizing a strongly convex R_n. Specifically, with $\alpha = 1/(2(cn+L))$, one can show that

$$\mathbb{E}[\|w_k - w_*\|_2^2] \leq \left(1 - \frac{c}{2(cn+L)}\right)^k \left(\|w_1 - w_*\|_2^2 + \frac{nD}{cn+L}\right),$$
$$\text{where} \;\; D := R_n(w_1) - \nabla R_n(w_*)^T(w_1 - w_*) - R_n(w_*).$$

Of course, attaining such a result requires knowledge of the strong convexity constant c and Lipschitz constant L. If c is not known, then the stepsize can instead be chosen to be $\alpha = 1/(3L)$ and a similar convergence result can be established; see [46].

Alternative initialization techniques could be used in practice, which may be more effective than evaluating all the gradients $\{\nabla f_i\}_{i=1}^n$ at the initial point. For example, one could perform one epoch of simple SG, or one can assimilate iterates one-by-one and compute g_k only using the gradients available up to that point.

One important drawback of Algorithm 5.2 is the need to store n stochastic gradient vectors, which would be prohibitive in many large-scale applications. Note, however, that if the component functions are of the form $f_i(w_k) = \hat{f}(x_i^T w_k)$, then

$$\nabla f_i(w_k) = \hat{f}'(x_i^T w_k) x_i.$$

That is, when the feature vectors $\{x_i\}$ are already available in storage, one need only store the scalar $\hat{f}'(x_i^T w_k)$ in order to construct $\nabla f_i(w_k)$ at a later iteration. Such a functional form of f_i occurs in logistic and least squares regression.

Algorithm 5.2 has its origins in the *stochastic average gradient* (SAG) algorithm [139, 90], where the stochastic direction is defined as

$$g_k \leftarrow \frac{1}{n}\left(\nabla f_j(w_k) - \nabla f_j(w_{[j]}) + \sum_{i=1}^n \nabla f_i(w_{[i]})\right). \tag{5.15}$$

Although this g_k is not an unbiased estimator of $\nabla R_n(w_k)$, the method enjoys a linear rate of convergence. One finds that the SAG algorithm is a randomized version of the Incremental Aggregated Gradient (IAG) method proposed in [17] and analyzed in [72], where the index j of the component function updated at every iteration is chosen cyclically. Interestingly, randomizing this choice yields good practical benefits.

5.3.3 Commentary

Although the gradient aggregation methods discussed in this section enjoy a faster rate of convergence than SG (i.e., linear vs. sublinear), they should not be regarded as clearly superior to SG. After all, following similar analysis as in §4.4, the computing time for SG can be shown to be $T(n, \epsilon) \sim \kappa^2/\epsilon$ with $\kappa := L/c$. (In fact, a computing time of κ/ϵ is often observed in practice.) On the other hand, the computing times for SVRG, SAGA, and SAG are

$$\mathcal{T}(n, \epsilon) \sim (n + \kappa) \log(1/\epsilon),$$

which grows with the number of examples n. Thus, following similar analysis as in §4.4, one finds that, for very large n, gradient aggregation methods are comparable to batch algorithms and therefore cannot beat SG in this regime. For example, if κ is close to 1, then SG is clearly more efficient since within a single epoch it reaches the optimal testing error [25]. On the other hand, there exists a regime with $\kappa \gg n$ in which gradient aggregation methods may be superior, and perhaps even easier to tune. At present, it is not known how useful gradient aggregation methods will prove to be in the future of large-scale machine learning. That being said, they certainly represent a class of optimization methods of interest due to their clever use of past information.

5.4 Iterate Averaging Methods

Since its inception, it has been observed that SG generates noisy iterate sequences that tend to oscillate around minimizers during the optimization process. Hence, a natural idea is to compute a corresponding sequence of *iterate averages* that would automatically possess less noisy behavior. Specifically, for minimizing a continuously differentiable F with unbiased gradient estimates, the idea is to employ the iteration

$$\begin{aligned} w_{k+1} &\leftarrow w_k - \alpha_k g(w_k, \xi_k) \\ \text{and} \quad \tilde{w}_{k+1} &\leftarrow \frac{1}{k+1} \sum_{j=1}^{k+1} w_j, \end{aligned} \tag{5.16}$$

where the averaged sequence $\{\tilde{w}_k\}$ has no effect on the computation of the SG iterate sequence $\{w_k\}$. Early hopes were that this auxiliary averaged sequence might possess better convergence properties than the SG iterates themselves. However, such improved behavior was found to be elusive when using classical stepsize sequences that diminished with a rate of $\mathcal{O}(1/k)$ [124].

A fundamental advancement in the use of iterate averaging came with the work of Polyak [125], which was subsequently advanced with Juditsky [126]; see also the work of Ruppert [136] and Nemirovski and Yudin [106]. Here, the idea remains to employ the iteration (5.16), but with stepsizes diminishing at a slower rate of $\mathcal{O}(1/(k^a))$ for some $a \in (\frac{1}{2}, 1)$. When minimizing strongly convex objectives, it follows from this choice that $\mathbb{E}[\|w_k - w_*\|_2^2] = \mathcal{O}(1/(k^a))$ while $\mathbb{E}[\|\tilde{w}_k - w_*\|_2^2] = \mathcal{O}(1/k)$. What is interesting, however, is that in certain cases this combination of *long steps* and *averaging* yields an optimal constant in the latter rate (for the iterate averages) in the sense that no rescaling of the steps—through multiplication with a positive definite matrix (see (6.2) in the next section)—can improve the asymptotic rate or constant. This shows that, due to averaging, the adverse effects caused by ill-conditioning disappear. (In this respect, the effect of averaging has been characterized as being similar to that of using a second-order method in which the Hessian approximations approach the Hessian of the objective at the minimizer [125, 22]; see §6.2.1.) This asymptotic behavior is difficult to achieve in practice, but is possible in some circumstances with careful selection of the stepsize sequence [161].

Iterate averaging has since been incorporated into various schemes in order to allow longer steps while maintaining desired rates of convergence. Examples include the *robust SA* and *mirror descent SA* methods presented in [105], as well as Nesterov's *primal-dual averaging* method proposed in [109, §6]. This latter method is notable in this section as it employs gradient aggregation and yields a $\mathcal{O}(1/k)$ rate of convergence for the averaged iterate sequence.

6 Second-Order Methods

In §5, we looked beyond classical SG to methods that are less affected by noise in the stochastic directions. Another manner to move beyond classical SG is to address the adverse effects of high nonlinearity and ill-conditioning of the objective function through the use of second-order information. As we shall see, these methods improve convergence rates of batch methods or the constants involved in the sublinear convergence rate of stochastic methods.

A common way to motivate second-order algorithms is to observe that first-order methods, such as SG and the full gradient method, are not *scale invariant*. Consider, for example, the full gradient iteration for minimizing a continuously differentiable function $F : \mathbb{R}^d \to \mathbb{R}$, namely

$$w_{k+1} \leftarrow w_k - \alpha_k \nabla F(w_k). \tag{6.1}$$

An alternative iteration is obtained by applying a full gradient approach after a linear transformation of variables, i.e., by considering $\min_{\tilde{w}} F(B\tilde{w})$ for some symmetric positive definite matrix B. The full gradient iteration for this problem has the form

$$\tilde{w}_{k+1} \leftarrow \tilde{w}_k - \alpha_k B \nabla F(B\tilde{w}_k),$$

which, after scaling by B and defining $\{w_k\} := \{B\tilde{w}_k\}$, corresponds to the iteration

$$w_{k+1} \leftarrow w_k - \alpha_k B^2 \nabla F(w_k). \tag{6.2}$$

Comparing (6.2) with (6.1), it is clear that the behavior of the algorithm will be different under this change of variables. For instance, when F is a strongly convex quadratic with unique minimizer w_*, the full gradient method (6.1) generally requires many iterations to approach the minimizer, but from any initial point w_1 the iteration (6.2) with $B = (\nabla^2 F(w_1))^{-1/2}$ and $\alpha_1 = 1$ will yield $w_2 = w_*$. These latter choices correspond to a single iteration of *Newton's method* [52]. In general, it is natural to seek transformations that perform well in theory and in practice.

Another motivation for second-order algorithms comes from the observation that each iteration of the form (6.1) or (6.2) chooses the subsequent iterate by first computing the minimizer of a second-order Taylor series approximation $q_k : \mathbb{R}^d \to \mathbb{R}$ of F at w_k, which has the form

$$q_k(w) = F(w_k) + \nabla F(w_k)^T (w - w_k) + \tfrac{1}{2}(w - w_k)^T B^{-2} (w - w_k). \tag{6.3}$$

The full gradient iteration corresponds to $B^{-2} = I$ while Newton's method corresponds to $B^{-2} = \nabla^2 F(w_k)$ (assuming this Hessian is positive definite). Thus, in general, a full gradient iteration works with a model that is only first-order accurate while Newton's method applies successive local re-scalings based on minimizing an exact second-order Taylor model of F at each iterate.

Deterministic (i.e., batch) methods are known to benefit from the use of second-order information; e.g., Newton's method achieves a quadratic rate of convergence if w_1 is sufficiently close to a strong minimizer [52]. On the other hand, stochastic methods like the SG method in §4 cannot achieve a convergence rate that is faster than sublinear, regardless of the choice of B; see [1, 104]. (More on this in §6.2.1.) Therefore, it is natural to ask: can there be a benefit to incorporating second-order information in stochastic methods? We address this question throughout this section by showing that the careful use of successive re-scalings based on (approximate) second-order derivatives can be beneficial between the stochastic and batch regimes.

We begin this section by considering a *Hessian-free Newton* method that employs exact second-order information, but in a judicious manner that exploits the stochastic nature of the objective function. We then describe methods that attempt to mimic the behavior of a Newton algorithm through first-order information computed over sequences of iterates; these include *quasi-Newton*, *Gauss-Newton*, and related algorithms that employ only diagonal re-scalings. We also discuss the *natural gradient* method, which defines a search direction in the space of realizable distributions, rather than in the space of the real parameter vector w. Whereas Newton's method is invariant to linear transformations of the variables, the natural gradient method is invariant with respect to more general invertible transformations.

We depict the methods of interest in this section on the downward axis illustrated in Figure 6.1. We use double-sided arrows for the methods that can be effective throughout the spectrum between the stochastic and batch regimes. Single-sided arrows are used for those methods that one might consider to be effective only with at least a moderate batch size in the stochastic gradient estimates. We explain these distinctions as we describe the methods.

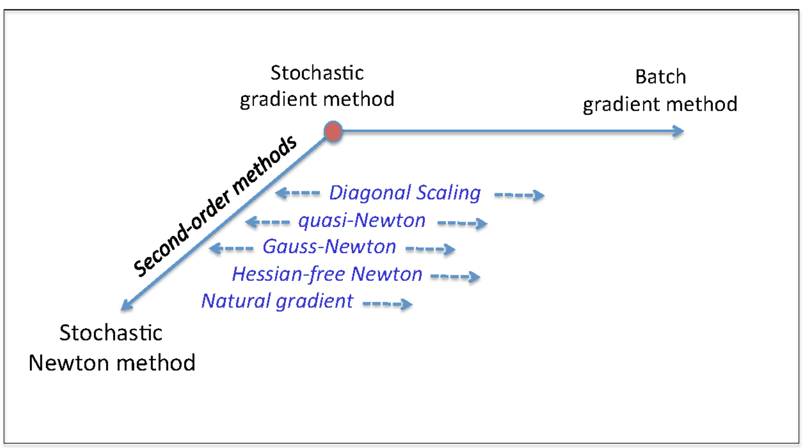

Fig. 6.1: View of the schematic from Figure 3.3 with a focus on second-order methods. The dotted arrows indicate the effective regime of each method: the first three methods can employ mini-batches of any size, whereas the last two methods are efficient only for moderate-to-large mini-batch sizes.

6.1 Hessian-Free Inexact Newton Methods

Due to its scale invariance properties and its ability to achieve a quadratic rate of convergence in the neighborhood of a strong local minimizer, Newton's method represents an ideal in terms of optimization algorithms. It does not scale well with the dimension d of the optimization problem, but there are variants that can scale well while also being able to deal with nonconvexity.

When minimizing a twice-continuously differentiable F, a Newton iteration is

$$w_{k+1} \leftarrow w_k + \alpha_k s_k, \tag{6.4a}$$

$$\text{where } s_k \text{ satisfies } \nabla^2 F(w_k) s_k = -\nabla F(w_k). \tag{6.4b}$$

This iteration demands much in terms of computation and storage. However, rather than solve the Newton system (6.4b) exactly through matrix factorization techniques, one can instead only solve it inexactly through an iterative approach such as the conjugate gradient (CG) method [69]. By ensuring that the linear solves are accurate enough, such an *inexact Newton-CG* method can enjoy a superlinear rate of convergence [47].

In fact, the computational benefits of inexact Newton-CG go beyond its ability to maintain classical convergence rate guarantees. Like many iterative linear system techniques, CG applied to (6.4b) does not require access to the Hessian itself, only Hessian-vector products [121]. It is in this sense that such a method may be called *Hessian-free*. This is ideal when such products can be coded directly without having to form an explicit Hessian, as Example 6.1 below demonstrates. Each product is at least as expensive as a gradient evaluation, but as long as the number of products—one per CG iteration—is not too large, the improved rate of convergence can compensate for the extra per-iteration work required over a simple full gradient method.

Example 6.1. *Consider the function of the parameter vector $w = (w_1, w_2)$ given by $F(w) = \exp(w_1 w_2)$. Let us define, for any $d \in \mathbb{R}^2$, the function*

$$\phi(w; d) = \nabla F(w)^T d = w_2 \exp(w_1 w_2) d_1 + w_1 \exp(w_1 w_2) d_2.$$

Computing the gradient of ϕ with respect to w, we have

$$\nabla_w \phi(w; d) = \nabla^2 F(w) d = \begin{bmatrix} w_2^2 \exp(w_1 w_2) d_1 + (\exp(w_1 w_2) + w_1 w_2 \exp(w_1 w_2)) d_2 \\ (\exp(w_1 w_2) + w_1 w_2 \exp(w_1 w_2)) d_1 + w_1^2 \exp(w_1 w_2) d_2 \end{bmatrix}.$$

We have thus obtained, for any $d \in \mathbb{R}^2$, a formula for computing $\nabla^2 F(w) d$ that does not require $\nabla^2 F(w)$ explicitly. Note that by storing the scalars $w_1 w_2$ and $\exp(w_1 w_2)$ from the evaluation of F, the additional costs of computing the gradient-vector and Hessian-vector products are small.

The idea illustrated in this example can be applied in general; e.g., see also Example 6.2 on page 54. For a smooth objective function F, one can compute $\nabla^2 F(w) d$ at a cost that is a small multiple of the cost of evaluating ∇F, and without forming the Hessian, which would require $\mathcal{O}(d^2)$ storage. The savings in computation come at the expense of storage of some additional quantities, as explained in Example 6.1.

In machine learning applications, including those involving multinomial logistic regression and deep neural networks, Hessian-vector products can be computed in this manner, and an inexact Newton-CG method can be applied. A concern is that, in certain cases, the cost of the CG iterations may render such a method uncompetitive with alternative approaches, such as an SG method or a limited memory BFGS method (see §6.2), which have small computational overhead. Interestingly, however, the structure of the risk measures (3.1) and (3.2) can be exploited so that the resulting method has lighter computational overheads, as described next.

6.1.1 Subsampled Hessian-Free Newton Methods

The motivation for the method we now describe stems from the observation that, in inexact Newton methods, the Hessian matrix need not be as accurate as the gradient to yield an effective iteration. Translated to the context of large-scale machine learning applications, this means that the iteration is more tolerant to noise in the Hessian estimate than it is to noise in the gradient estimate.

Based on this idea, the technique we state here employs a smaller sample for defining the Hessian than for the stochastic gradient estimate. Following similar notation as introduced in §5.2, let the stochastic gradient estimate be

$$\nabla f_{\mathcal{S}_k}(w_k;\xi_k) = \frac{1}{|\mathcal{S}_k|}\sum_{i\in\mathcal{S}_k}\nabla f(w_k;\xi_{k,i})$$

and let the stochastic Hessian estimate be

$$\nabla^2 f_{\mathcal{S}_k^H}(w_k;\xi_k^H) = \frac{1}{|\mathcal{S}_k^H|}\sum_{i\in\mathcal{S}_k^H}\nabla^2 f(w_k;\xi_{k,i}), \tag{6.5}$$

where $\mathcal{S}_k^H$ is conditionally uncorrelated with $\mathcal{S}_k$ given w_k. If one chooses the *subsample* size $|\mathcal{S}_k^H|$ small enough, then the cost of each product involving the Hessian approximation can be reduced significantly, thus reducing the cost of each CG iteration. On the other hand, one should choose $|\mathcal{S}_k^H|$ large enough so that the curvature information captured through the Hessian-vector products is productive. If done appropriately, *Hessian subsampling* is robust and effective [2, 29, 122, 132]. An inexact Newton method that incorporates this techniques is outlined in Algorithm 6.1. The algorithm is stated with a backtracking (Armijo) line search [114], though other stepsize-selection techniques could be considered as well.

Algorithm 6.1 Subsampled Hessian-Free Inexact Newton Method

1: Choose an initial iterate w_1.
2: Choose constants $\rho\in(0,1)$, $\gamma\in(0,1)$, $\eta\in(0,1)$, and $\max_{cg}\in\mathbb{N}$.
3: **for** $k=1,2,\dots$ **do**
4: Generate realizations of ξ_k and ξ_k^H corresponding to $\mathcal{S}_k$ and $\mathcal{S}_k^H$.
5: Compute s_k by applying Hessian-free CG to solve

$$\nabla^2 f_{\mathcal{S}_k^H}(w_k;\xi_k^H)s = -\nabla f_{\mathcal{S}_k}(w_k;\xi_k) \tag{6.6}$$

until $\max_{cg}$ iterations have been performed or a trial solution yields

$$\|r_k\|_2 := \|\nabla^2 f_{\mathcal{S}_k^H}(w_k;\xi_k^H)s + \nabla f_{\mathcal{S}_k}(w_k;\xi_k)\|_2 \le \rho\|\nabla f_{\mathcal{S}_k}(w_k;\xi_k)\|_2.$$

6: Set $w_{k+1}\leftarrow w_k+\alpha_k s_k$, where $\alpha_k\in\{\gamma^0,\gamma^1,\gamma^2,\dots\}$ is the largest element with

$$f_{\mathcal{S}_k}(w_{k+1};\xi_k) \le f_{\mathcal{S}_k}(w_k;\xi_k) + \eta\alpha_k\nabla f_{\mathcal{S}_k}(w_k;\xi_k)^T s_k. \tag{6.7}$$

7: **end for**

As previously mentioned, the (subsampled) Hessian-vector products required in Algorithm 6.1 can be computed efficiently in the context of many machine learning applications. For instance, one such case in the following.

Example 6.2. *Consider a binary classification problem where the training function is given by the logistic loss with an ℓ_2-norm regularization parameterized by $\lambda > 0$:*

$$R_n(w) = \frac{1}{n}\sum_{i=1}^{n} \log(1+\exp(-y_i w^T x_i)) + \frac{\lambda}{2}\|w\|^2. \tag{6.8}$$

A (subsampled) Hessian-vector product can be computed efficiently by observing that

$$\nabla^2 f_{\mathcal{S}_k^H}(w_k;\xi_k^H)d = \frac{1}{|\mathcal{S}_k^H|}\sum_{i\in\mathcal{S}_k^H} \frac{\exp(-y_i w^T x_i)}{(1+\exp(-y_i w^T x_i))^2}(x_i^T d)x_i + \lambda d.$$

To quantify the step computation cost in an inexact Newton-CG framework such as Algorithm 6.1, let g_{cost} be the cost of computing a gradient estimate $\nabla f_{\mathcal{S}_k}(w_k;\xi_k)$ and $\text{factor}\times\text{g}_{cost}$ denote the cost of one Hessian-vector product. If the maximum number of CG iterations, $\max_{cg}$, is performed for every outer iteration, then the step computation cost in Algorithm 6.1 is

$$\max_{cg} \times \text{factor} \times \text{g}_{cost} + \text{g}_{cost}.$$

In a deterministic inexact Newton-CG method for minimizing the empirical risk R_n, i.e., when $|\mathcal{S}_k^H| = |\mathcal{S}_k| = n$ for all $k \in \mathbb{N}$, the factor is at least 1 and $\max_{cg}$ would typically be chosen as 5, 20, or more, leading to an iteration that is many times the cost of an SG iteration. However, in a stochastic framework using Hessian sub-sampling, the factor can be chosen to be sufficiently small such that $\max_{cg} \times \text{factor} \approx 1$, leading to a per-iteration cost proportional to that of SG.

Implicit in this discussion is the assumption that the gradient sample size $|\mathcal{S}_k|$ is large enough so that taking subsamples for the Hessian estimate is sensible. If, by contrast, the algorithm were to operate in the stochastic regime of SG where $|\mathcal{S}_k|$ is small and gradients are very noisy, then it may be necessary to choose $|\mathcal{S}_k^H| > |\mathcal{S}_k|$ so that Hessian approximations do not corrupt the step. In such circumstances, the method would be far less attractive than SG. Therefore, the subsampled Hessian-free Newton method outlined here is only recommended when $\mathcal{S}_k$ is large. This is why, in Figure 6.1, the Hessian-free Newton method is illustrated only with an arrow to the right, i.e., in the direction of larger sample sizes.

Convergence of Algorithm 6.1 is easy to establish when minimizing a strongly convex empirical risk measure $F = R_n$ when $\mathcal{S}_k \leftarrow \{1,\dots,n\}$ for all $k \in \mathbb{N}$, i.e., when full gradients are always used. In this case, a benefit of employing CG to solve (6.6) is that it immediately improves upon the direction employed in a steepest descent iteration. Specifically, when initialized at zero, it produces in its first iteration a scalar multiple of the steepest descent direction $-\nabla F(w_k)$, and further iterations monotonically improve upon this step (in terms of minimizing a quadratic model of the form in (6.3)) until the Newton step is obtained, which is done so in at most d iterations of CG (in exact arithmetic). Therefore, by using any number of CG iterations, convergence can be established using standard techniques to choose the stepsize α_k [114]. When exact Hessians are also used, the rate of convergence can be controlled through the accuracy with which the systems (6.4b) are solved. Defining $r_k := \nabla^2 F(w_k)s_k + \nabla F(w_k)$ for all $k \in \mathbb{N}$, the iteration can enjoy a linear, superlinear, or quadratic rate of convergence by controlling $\|r_k\|_2$, where for the superlinear rates one must have $\{\|r_k\|_2/\|\nabla F(w_k)\|_2\} \to 0$ [47].

When the Hessians are subsampled (i.e., $\mathcal{S}_k^H \subset \mathcal{S}_k$ for all $k \in \mathbb{N}$), it has not been shown that the rate of convergence is faster than linear; nevertheless, the reduction in the number of iterations required to produce a good approximate solution is often significantly lower than if no Hessian information is used in the algorithm.

6.1.2 Dealing with Nonconvexity

Hessian-free Newton methods are routinely applied for the solution of nonconvex problems. In such cases, it is common to employ a *trust region* [37] instead of a line search and to introduce an additional condition in Step 5 of Algorithm 6.1: terminate CG if a candidate solution s_k is a direction of negative curvature, i.e., $s_k^T \nabla^2 f_{\mathcal{S}_k^H}(w_k; \xi_k^H) s_k < 0$ [149]. A number of more sophisticated strategies have been proposed throughout the years with some success, but none have proved to be totally satisfactory or universally accepted.

Instead of coping with indefiniteness, one can focus on strategies for ensuring positive (semi)definite Hessian approximations. One of the most attractive ways of doing this in the context of machine learning is to employ a (subsampled) Gauss-Newton approximation to the Hessian, which is a matrix of the form

$$G_{\mathcal{S}_k^H}(w_k; \xi_k^H) = \frac{1}{|\mathcal{S}_k^H|} \sum_{i \in \mathcal{S}_k^H} J_h(w_k, \xi_{k,i})^T H_\ell(w_k, \xi_{k,i}) J_h(w_k, \xi_{k,i}). \tag{6.9}$$

Here, the matrix J_h captures the stochastic gradient information for the prediction function $h(x; w)$, whereas the matrix H_ℓ only captures the second order information for the (convex) loss function $\ell(h, y)$; see §6.3 for a detailed explanation. As before, one can directly code the product of this matrix times a vector without forming the matrix components explicitly. This approach has been applied successfully in the training of deep neural networks [12, 100].

We mention in passing that there has been much discussion about the role that negative curvature and saddle points play in the optimization of deep neural networks; see e.g., [44, 70, 35]. Numerical tests designed to probe the geometry of the objective function in the neighborhood of a minimizer when training a deep neural network have shown the presence of negative curvature. It is believed that the inherent stochasticity of the SG method allows it to navigate efficiently through this complex landscape, but it is not known whether classical techniques to avoid approaching saddle points will prove to be successful for either batch or stochastic methods.

6.2 Stochastic Quasi-Newton Methods

One of the most important developments in the field of nonlinear optimization came with the advent of *quasi-Newton* methods. These methods construct approximations to the Hessian using only gradient information, and are applicable for convex and nonconvex problems. Versions that scale well with the number of variables, such as *limited memory* methods, have proved to be effective in a wide range of applications where the number of variables can be in the millions. It is therefore natural to ask whether quasi-Newton methods can be extended to the stochastic setting arising in machine learning. Before we embark on this discussion, let us review the basic principles underlying quasi-Newton methods, focusing on the most popular scheme, namely BFGS [27, 60, 68, 146].

In the spirit of Newton's method (6.4), the BFGS iteration for minimizing a twice continuously differentiable function F has the form

$$w_{k+1} \leftarrow w_k - \alpha_k H_k \nabla F(w_k), \tag{6.10}$$

where H_k is a symmetric positive definite approximation of $(\nabla^2 F(w_k))^{-1}$. This form of the iteration is consistent with (6.4), but the signifying feature of a quasi-Newton scheme is that the sequence

$\{H_k\}$ is updated dynamically by the algorithm rather than through a second-order derivative computation at each iterate. Specifically, in the BFGS method, the new inverse Hessian approximation is obtained by defining the iterate and gradient displacements

$$s_k := w_{k+1} - w_k \quad \text{and} \quad v_k := \nabla F(w_{k+1}) - \nabla F(w_k),$$

then setting

$$H_{k+1} \leftarrow \left(I - \frac{v_k s_k^T}{s_k^T v_k}\right)^T H_k \left(I - \frac{v_k s_k^T}{s_k^T v_k}\right) + \frac{s_k s_k^T}{s_k^T v_k}. \tag{6.11}$$

One important aspect of this update is that it ensures that the secant equation $H_{k+1}^{-1} s_k = v_k$ holds, meaning that a second-order Taylor expansion is satisfied along the most recent displacement (though not necessarily along other directions).

A remarkable fact about the BFGS iteration (6.10)–(6.11) is that it enjoys a local superlinear rate of convergence [51], and this with only first-order information and without the need for any linear system solves (which are required by Newton's method for it to be quadratically convergent). However, a number of issues need to be addressed to have an effective method in practice. For one thing, the update (6.11) yields dense matrices, even when the exact Hessians are sparse, restricting its use to small and midsize problems. A common solution for this is to employ a *limited memory* scheme, leading to a method such as L-BFGS [97, 113]. A key feature of this approach is that the matrices $\{H_k\}$ need not be formed explicitly; instead, each product of the form $H_k \nabla F(w_k)$ can be computed using a formula that only requires recent elements of the sequence of displacement pairs $\{(s_k, v_k)\}$ that have been saved in storage. Such an approach incurs per-iteration costs of order $\mathcal{O}(d)$, and delivers practical performance that is significantly better than an full gradient iteration, though the rate of convergence is only provably linear.

6.2.1 Deterministic to Stochastic

Let us consider the extension of a quasi-Newton approach from the deterministic to the stochastic setting arising in machine learning. The iteration now takes the form

$$w_{k+1} \leftarrow w_k - \alpha_k H_k g(w_k, \xi_k). \tag{6.12}$$

Since we are interested in large-scale problems, we assume that (6.12) implements an L-BFGS scheme, which avoids the explicit construction of H_k. A number of questions arise when considering (6.12). We list them now with some proposed solutions.

Theoretical Limitations The convergence rate of a stochastic iteration such as (6.12) cannot be faster than sublinear [1]. Given that SG also has a sublinear rate of convergence, what benefit, if any, could come from incorporating H_k in (6.12)? This is an important question. As it happens, one can see a benefit of H_k in terms of the *constant* that appears in the sublinear rate. Recall that for the SG method (Algorithm 4.1), the constant depends on L/c, which in turn depends on the conditioning of $\{\nabla^2 F(w_k)\}$. This is typical of first-order methods. In contrast, one can show [25] that if the sequence of Hessian approximations in (6.12) satisfies $\{H_k\} \to \nabla^2 F(w_*)^{-1}$, then the constant is independent of the conditioning of the Hessian. Although constructing Hessian approximations with this property might not be viable in practice, this fact suggests that stochastic quasi-Newton methods could be better equipped to cope with ill-conditioning than SG.

Additional Per-Iteration Costs The SG iteration is very inexpensive, requiring only the evaluation of $g(w_k, \xi_k)$. The iteration (6.12), on the other hand, also requires the product $H_k g(w_k, \xi_k)$, which is known to require $4md$ operations where m is the *memory* in the L-BFGS updating scheme. Assuming for concreteness that the cost of evaluating $g(w_k, \xi_k)$ is exactly d operations (using only one sample) and that the memory parameter is set to the typical value of $m = 5$, one finds that the stochastic quasi-Newton method is 20 times more expensive than SG. Can the iteration (6.12) yield fast enough progress as to offset this additional per-iteration cost? To address this question, one need only observe that the calculation just mentioned focuses on the gradient $g(w_k, \xi_k)$ being based on a single sample. However, when employing mini-batch gradient estimates, the additional cost of the iteration (6.12) is only marginal. (Mini-batches of size 256 are common in practice.) The use of mini-batches may therefore be considered essential when one contemplates the use of a stochastic quasi-Newton method. This mini-batch need not be large, as in the Hessian-free Newton method discussed in the previous section, but it should not be less than, say, 20 or 50 in light of the additional costs of computing the matrix-vector products.

Conditioning of the Scaling Matrices The BFGS formula (6.11) for updating H_k involves differences in gradient estimates computed in consecutive iterations. In stochastic settings, the gradients $\{g(w_k, \xi_k)\}$ are noisy estimates of $\{\nabla F(w_k)\}$. This can cause the updating process to yield poor curvature estimates, which may have a detrimental rather than beneficial effect on the quality of the computed steps. Since BFGS, like all quasi-Newton schemes, is an *overwriting* process, the effects of even a single bad update may linger for numerous iterations. How could such effects be avoided in the stochastic regime? There have been various proposals to avoid differencing noisy gradient estimates. One possibility is to employ the same sample when computing gradient differences [18, 142]. An alternative approach that allows greater freedom in the choice of the stochastic gradient is to *decouple* the step computation (6.12) and the Hessian update. In this manner, one can employ a larger sample, if necessary, when computing the gradient displacement vector. We discuss these ideas further in §6.2.2.

It is worthwhile to note that if the gradient estimate $g(w_k, \xi_k)$ does not have high variance, then standard BFGS updating can be applied without concerns. Therefore, in the rest of this section, we focus on algorithms that employ noisy gradient estimates in the step computation (6.12). This means, e.g., that we are not considering the potential to tie the method with noise reduction techniques described in §5, though such an idea is natural and could be effective in practice.

6.2.2 Algorithms

A straightforward adaptation of L-BFGS involves only the replacement of deterministic gradients with stochastic gradients throughout the iterative process. The displacement pairs might then be defined as

$$s_k := w_{k+1} - w_k \quad \text{and} \quad v_k := \nabla f_{\mathcal{S}_k}(w_{k+1}, \xi_k) - \nabla f_{\mathcal{S}_k}(w_k, \xi_k). \tag{6.13}$$

Note the use of the same seed ξ_k in the two gradient estimates, in order to address the issues related to noise mentioned above. If each f_i is strongly convex, then $s_k^T v_k > 0$, and positive definiteness of the updates is also maintained. Such an approach is sometimes referred to as *online L-BFGS* [142, 103].

One disadvantage of this method is the need to compute two, as opposed to only one, gradient estimate per iteration: one to compute the gradient displacement (namely, $g(w_{k+1}, \xi_k)$) and another (namely, $g(w_{k+1}, \xi_{k+1})$) to compute the subsequent step. This is not too onerous, at least as long as the per-iteration improvement outweighs the extra per-iteration cost. A more worrisome feature is that updating the inverse Hessian approximations with *every* step may not be warranted, and may even be detrimental when the gradient displacement is based on a small sample, since it could easily represent a poor approximation of the action of the true Hessian of F.

An alternative strategy, which might better represent the action of the true Hessian even when $g(w_k, \xi_k)$ has high variance, is to employ an alternative v_k. In particular, since $\nabla F(w_{k+1}) - \nabla F(w_k) \approx \nabla^2 F(w_k)(w_{k+1} - w_k)$, one could define

$$v_k := \nabla^2 f_{\mathcal{S}_k^H}(w_k; \xi_k^H) s_k, \tag{6.14}$$

where $\nabla^2 f_{\mathcal{S}_k^H}(w_k; \xi_k^H)$ is a subsampled Hessian and $|\mathcal{S}_k^H|$ is large enough to provide useful curvature information. As in the case of Hessian-free Newton from §6.1.1, the product (6.14) can be performed without explicitly constructing $\nabla^2 f_{\mathcal{S}_k^H}(w_k; \xi_k^H)$.

Regardless of the definition of v_k, when $|\mathcal{S}_k^H|$ is much larger than $|\mathcal{S}_k|$, the cost of quasi-Newton updating is excessive due to the cost of computing v_k. To address this issue, the computation of v_k can be performed only after a sequence of iterations, so as to amortize costs. This leads to the idea of decoupling the step computation from the quasi-Newton update. This approach, which we refer to for convenience as SQN, performs a sequence of iterations of (6.12) with H_k fixed, then computes a new displacement pair (s_k, v_k) with s_k defined as in (6.13) and v_k set using one of the strategies outlined above. This pair replaces one of the old pairs in storage, which in turn defines the limited memory BFGS step.

To formalize all of these alternatives, we state the general stochastic quasi-Newton method presented as Algorithm 6.2, with some notation borrowed from Algorithm 6.1. In the method, the step computation is based on a collection of m displacement pairs $\mathcal{P} = \{s_j, v_j\}$ in storage and the current stochastic gradient $\nabla f_{\mathcal{S}_k}(w_k; \xi_k)$, where the matrix-vector product in (6.12) can be computed through a *two-loop recursion* [113, 114]. To demonstrate the generality of the method, we note that the *online L-BFGS* method sets $\mathcal{S}_k^H \leftarrow \mathcal{S}_k$ and `update pairs = true` in every iteration. In *SQN* using (6.14), on the other hand, $|\mathcal{S}_k^H|$ should be chosen larger than $|\mathcal{S}_k|$ and one sets `update pairs = true` only every, say, 10 or 20 iterations.

To guarantee that the BFGS update is well defined, each displacement pair (s_j, v_j) must satisfy $s_j^T v_j > 0$. In deterministic optimization, this issue is commonly addressed by either performing a line search (involving exact gradient computations) or modifying the displacement vectors (e.g., through *damping*) so that $s_j^T v_j > 0$, in which case one does ensure that (6.11) maintains positive definite approximations. However, these mechanisms have not been fully developed in the stochastic regime when exact gradient information is unavailable and the gradient displacement vectors are noisy. Simple ways to overcome these difficulties is to replace the Hessian matrix with a Gauss-Newton approximation or to introduce a combination of damping and regularization (say, through the addition of simple positive definite matrices).

There remains much to be explored in terms of stochastic quasi-Newton methods for machine learning applications. Experience has shown that some gains in performance can be achieved, but the full potential of the quasi-Newton schemes discussed above (and potentially others) is not yet known.

Algorithm 6.2 Stochastic Quasi-Newton Framework

```
Choose an initial iterate w_1 and initialize P ← ∅.
Choose a constant m ∈ ℕ.
Choose a stepsize sequence {α_k} ⊂ ℝ_{++}.
for k = 1, 2, ..., do
    Generate realizations of ξ_k and ξ_k^H corresponding to S_k and S_k^H.
    Compute ŝ_k = H_k g(w_k, ξ_k) using the two-loop recursion based on the set P.
    Set s_k ← −α_k ŝ_k.
    Set w_{k+1} ← w_k + s_k.
    if update pairs then
        Compute s_k and v_k (based on the sample S_k^H).
        Add the new displacement pair (s_k, v_k) to P.
        If |P| > m, then remove eldest pair from P.
    end if
end for
```

6.3 Gauss-Newton Methods

The Gauss-Newton method is a classical approach for nonlinear least squares, i.e., minimization problems in which the objective function is a sum of squares. This method readily applies for optimization problems arising in machine learning involving a least squares loss function, but the idea generalizes for other popular loss functions as well. The primary advantage of Gauss-Newton is that it constructs an approximation to the Hessian using only first-order information, and this approximation is guaranteed to be positive semidefinite, even when the full Hessian itself may be indefinite. The price to pay for this convenient representation is that it ignores second-order interactions between elements of the parameter vector w, which might mean a loss of curvature information that could be useful for the optimization process.

Classical Gauss-Newton Let us introduce the classical Gauss-Newton approach by considering a situation in which, for a given input-output pair (x, y), the loss incurred by a parameter vector w is measured via a squared norm discrepancy between $h(x; w) \in \mathbb{R}^d$ and $y \in \mathbb{R}^d$. Representing the input-output pair being chosen randomly via the subscript ξ, we may thus write

$$f(w; \xi) = \ell(h(x_\xi; w), y_\xi) = \tfrac{1}{2}\|h(x_\xi; w) - y_\xi\|_2^2.$$

Writing a second-order Taylor series model of this function in the vicinity of parameter vector w_k would involve its gradient and Hessian at w_k, and minimizing the resulting model (recall (6.3)) would lead to a Newton iteration. Alternatively, a Gauss-Newton approximation of the function is obtained by making an affine approximation of the prediction function inside the quadratic loss function. Letting $J_h(\cdot; \xi)$ represent the Jacobian of $h(x_\xi; \cdot)$ with respect to w, we have the approximation

$$h(x_\xi; w) \approx h(x_\xi; w_k) + J_h(w_k; \xi)(w - w_k),$$

which leads to

$$\begin{aligned}
f(w;\xi) &\approx \tfrac{1}{2}\|h(x_\xi;w_k) + J_h(w_k;\xi)(w-w_k) - y_\xi\|_2^2 \\
&= \tfrac{1}{2}\|h(x_\xi;w_k) - y_\xi\|_2^2 + (h(x_\xi;w_k)-y_\xi)^T J_h(w_k;\xi)(w-w_k) \\
&\quad + \tfrac{1}{2}(w-w_k)^T J_h(w_k;\xi)^T J_h(w_k;\xi)(w-w_k).
\end{aligned}$$

In fact, this approximation is similar to a second-order Taylor series model, except that the terms involving the second derivatives of the prediction function h with respect to the parameter vector have been dropped. The remaining second-order terms are those resulting from the positive curvature of the quadratic loss ℓ. This leads to replacing the subsampled Hessian matrix (6.5) by the Gauss-Newton matrix

$$G_{\mathcal{S}_k^H}(w_k;\xi_k^H) = \frac{1}{|\mathcal{S}_k^H|}\sum_{i\in\mathcal{S}_k^H} J_h(w_k;\xi_{k,i})^T J_h(w_k;\xi_{k,i})\,. \tag{6.15}$$

Since the Gauss-Newton matrix only differs from the true Hessian by terms that involve the factors $h(x_\xi;w_k)-y_\xi$, these two matrices are the same when the loss is equal to zero, i.e., when $h(x_\xi;w_k) = y_\xi$.

A challenge in the application of a Gauss-Newton scheme is that the Gauss-Newton matrix is often singular or nearly singular. In practice, this is typically handled by regularizing it by adding to it a positive multiple of the identity matrix. For least-squares loss functions, the inexact Hessian-free Newton methods of §6.1 and the stochastic quasi-Newton methods of §6.2 with gradient displacement vectors defined as in (6.14) can be applied with (regularized) Gauss-Newton approximations. This has the benefit that the scaling matrices are guaranteed to be positive definite.

The computational cost of the Gauss-Newton method depends on the dimensionality of the prediction function. When the prediction function is scalar-valued, the Jacobian matrix J_h is a single row whose elements are already being computed as an intermediate step in the computation of the stochastic gradient $\nabla f(w;\xi)$. However, this is no longer true when the dimensionality is larger than one since then computing the stochastic gradient vector $\nabla f(w;\xi)$ does not usually require the explicit computation of all rows of the Jacobian matrix. This happens, for instance, in deep neural networks when one uses back propagation [134, 135].

Generalized Gauss-Newton Gauss-Newton ideas can also be generalized for other standard loss functions [141]. To illustrate, let us consider a slightly more general situation in which loss is measured by a composition of an arbitrary convex loss function $\ell(h,y)$ and a prediction function $h(x;w)$. Combining the affine approximation of the prediction function $h(x_\xi;w)$ with a second order Taylor expansion of the loss function ℓ leads to the generalized Gauss-Newton matrix

$$G_{\mathcal{S}_k^H}(w_k;\xi_k^H) = \frac{1}{|\mathcal{S}_k^H|}\sum_{i\in\mathcal{S}_k^H} J_h(w_k;\xi_{k,i})^T H_\ell(w_k;\xi_{k,i})\, J_h(w_k;\xi_{k,i}) \tag{6.16}$$

(recall (6.9)), where $H_\ell(w_k;\xi) = \frac{\partial^2\ell}{\partial h^2}(h(x_\xi;w_k),y_\xi)$ captures the curvature of the loss function ℓ. This can be seen as a generalization of (6.15) in which $H_\ell = I$.

When training a deep neural network, one may exploit this generalized strategy by redefining ℓ and h so that as much as possible of the network's computation is formally performed by ℓ rather than by h. If this can be done in such a way that convexity of ℓ is maintained, then one can faithfully capture second-order terms for ℓ using the generalized Gauss-Newton scheme. Interestingly, in many useful situations, this strategy gives simpler and more elegant expressions for H_ℓ. For instance, probability estimation problems often reduce to using logarithmic losses of the form $f(w;\xi) = -\log(h(x_\xi; w))$. The generalized Gauss-Newton matrix then reduces to

$$\begin{aligned} G_{\mathcal{S}_k^H}(w_k;\xi_k^H) &= \frac{1}{|\mathcal{S}_k^H|} \sum_{i\in\mathcal{S}_k^H} J_h(w_k;\xi_{k,i})^T \frac{1}{h(w;\xi_{k,i})^2} J_h(w_k;\xi_{k,i}) \\ &= \frac{1}{|\mathcal{S}_k^H|} \sum_{i\in\mathcal{S}_k^H} \nabla f(w;\xi_{k,i}) \nabla f(w;\xi_{k,i})^T, \end{aligned} \tag{6.17}$$

which does not require explicit computation of the Jacobian J_h.

6.4 Natural Gradient Method

We have seen that Newton's method is invariant to *linear* transformations of the parameter vector w. By contrast, the natural gradient method [5, 6] aims to be invariant with respect to all differentiable and invertible transformations. The essential idea consists of formulating the gradient descent algorithm in the space of prediction functions rather than specific parameters. Of course, the actual computation takes place with respect to the parameters, but accounts for the anisotropic relation between the parameters and the decision function. That is, in parameter space, the natural gradient algorithm will move the parameters more quickly along directions that have a small impact on the decision function, and more cautiously along directions that have a large impact on the decision function.

We remark at the outset that many authors [119, 99] propose *quasi-natural-gradient* methods that are strikingly similar to the *quasi-Newton* methods described in §6.2. The natural gradient approach therefore offers a different justification for these algorithms, one that involves qualitatively different approximations. It should also be noted that research on the design of methods inspired by the natural gradient is ongoing and may lead to markedly different algorithms [33, 76, 101].

Information Geometry In order to directly formulate the gradient descent in the space of prediction functions, we must elucidate the geometry of this space. Amari's work on information geometry [6] demonstrates this for parametric density estimation. The space $\mathcal{H}$ of prediction functions for such a problem is a family of densities $h_w(x)$ parametrized by $w \in \mathcal{W}$ and satisfying the normalization condition

$$\int h_w(x)\,dx = 1 \quad \text{for all} \quad w \in \mathcal{W}.$$

Assuming sufficient regularity, the derivatives of such densities satisfy the identity

$$\forall t > 0 \qquad \int \frac{\partial^t h_w(x)}{\partial w^t}\,dx = \frac{\partial^t}{\partial w^t} \int h_w(x)\,dx = \frac{\partial^t 1}{\partial w^t} = 0. \tag{6.18}$$

To elucidate the geometry of the space $\mathcal{H}$, we seek to quantify how the density h_w changes when one adds a small quantity δw to its parameter. We achieve this in a statistically meaningful way by observing the Kullback-Leibler (KL) divergence

$$D_{KL}(h_w \| h_{w+\delta w}) = \mathbb{E}_{h_w}\left[\log\left(\frac{h_w(x)}{h_{w+\delta w}(x)}\right)\right], \tag{6.19}$$

where $\mathbb{E}_{h_w}$ denotes the expectation with respect to the distribution h_w. Note that (6.19) only depends on the values of the two density functions h_w and $h_{w+\delta w}$ and therefore is invariant with respect to any invertible transformation of the parameter w. Approximating the divergence with a second-order Taylor expansion, one obtains

$$\begin{aligned} D_{KL}(h_w \| h_{w+\delta w}) &= \mathbb{E}_{h_w}[\log(h_w(x)) - \log(h_{w+\delta w}(x))] \\ &\approx -\delta w^T \mathbb{E}_{h_w}\left[\frac{\partial \log(h_w(x))}{\partial w}\right] - \tfrac{1}{2}\delta w^T \mathbb{E}_{h_w}\left[\frac{\partial^2 \log(h_w(x))}{\partial w^2}\right]\delta w, \end{aligned}$$

which, after observing that (6.18) implies that the first-order term is null, yields

$$D_{KL}(h_w \| h_{w+\delta w}) \approx \tfrac{1}{2}\delta w^T G(w) \delta w. \tag{6.20}$$

This is a quadratic form defined by the *Fisher information* matrix

$$G(w) := -\mathbb{E}_{h_w}\left[\frac{\partial^2 \log(h_w(x))}{\partial w^2}\right] = -\mathbb{E}_{h_w}\left[\left(\frac{\partial \log(h_w(x))}{\partial w}\right)\left(\frac{\partial \log(h_w(x))}{\partial w}\right)^T\right], \tag{6.21}$$

where the latter equality follows again from (6.18). The second form of $G(w)$ is often preferred because it makes clear that the Fisher information matrix $G(w)$ is symmetric and always positive semidefinite, though not necessarily positive definite.

The relation (6.20) means that the KL divergence behaves locally like a norm associated with $G(w)$. Therefore, every small region of $\mathcal{H}$ looks like a small region of a Euclidean space. However, as we traverse larger regions of $\mathcal{H}$, we cannot ignore that the matrix $G(w)$ changes. Such a construction defines a *Riemannian geometry*.[7]

Suppose, for instance, that we move along a smooth path connecting two densities, call them h_{w_0} and h_{w_1}. A parametric representation of the path can be given by a differentiable function, for which we define

$$\phi : t \in [0,1] \mapsto \phi(t) \in \mathcal{W} \quad \text{with} \quad \phi(0) = w_0 \quad \text{and} \quad \phi(1) = w_1.$$

We can compute the length of the path by viewing it as a sequence of infinitesimal segments $[\phi(t), \phi(t+dt)]$ whose length is given by (6.20), i.e., the total length is

$$D_\phi = \int_0^1 \sqrt{\left(\frac{d\phi}{dt}(t)\right)^T G(\phi(t)) \left(\frac{d\phi}{dt}(t)\right)}\, dt.$$

An important tool for the study of Riemannian geometries is the characterization of its *geodesics*, i.e., the shortest paths connecting two points. In a Euclidean space, the shortest path between two points is always the straight line segment connecting them. In a Riemannian space, on the other hand, the shortest path between two points can be curved and does not need to be unique. Such considerations are relevant to optimization since every iterative optimization algorithm can be viewed as attempting to follow a particular path connecting the initial point w_0 to the optimum w_*. In particular, following the shortest path is attractive because it means that the algorithm reaches the optimum after making the fewest number of changes to the optimization variables, hopefully requiring the least amount of computation.

[7] The objective of information geometry [6] is to exploit the Riemannian structure of parametric families of density functions to gain geometrical insights on the fundamental statistical phenomena. The natural gradient algorithm is only a particular aspect of this broader goal [5].

Natural Gradient Let us now assume that the space $\mathcal{H}$ of prediction functions $\{h_w : w \in \mathcal{W}\}$ has a Riemannian geometry locally described by an identity of the form (6.20). We seek an algorithm that minimizes a functional $F : h_w \in \mathcal{H} \mapsto F(h_w) = F(w) \in \mathbb{R}$ and is invariant with respect to differentiable invertible transformations of the parameters represented by the vector w.

Each iteration of a typical iterative optimization algorithm computes a new iterate $h_{w_{k+1}}$ on the basis of information pertaining to the current iterate h_{w_k}. Since we can only expect this information to be valid in a small region surrounding h_{w_k}, we restrict our attention to algorithms that make a step from h_{w_k} to $h_{w_{k+1}}$ of some small length $\eta_k > 0$. The number of iterations needed to reach the optimum then depends directly on the length of the path followed by the algorithm, which is desired to be as short as possible. Unfortunately, it is rarely possible to exactly follow a geodesic using only local information. We can, however, formulate the greedy strategy that

$$h_{w_{k+1}} = \arg\min_{h \in \mathcal{H}} \; F(h) \quad \text{s.t.} \quad D(h_{w_k} \| h) \le \eta_k^2, \tag{6.22}$$

and use (6.20) to reformulate this problem in terms of the parameters:

$$w_{k+1} = \arg\min_{w \in \mathcal{W}} \; F(w) \quad \text{s.t.} \quad \tfrac{1}{2}(w - w_k)^T G(w_k)\,(w - w_k) \le \eta_k^2\,. \tag{6.23}$$

The customary derivation of the natural gradient algorithm handles the constraint in (6.23) using a Lagrangian formulation with Lagrange multiplier $1/\alpha_k$. In addition, since η_k is assumed small, it replaces $F(w)$ in (6.23) by the first-order approximation $F(w_k) + \nabla F(w_k)^T(w - w_k)$. These two choices lead to the expression

$$w_{k+1} = \arg\min_{w \in \mathcal{W}} \; \nabla F(w_k)^T(w - w_k) + \frac{1}{2\alpha_k}(w - w_k)^T G(w_k)(w - w_k),$$

the optimization of the right-hand side of which leads to the natural gradient iteration

$$w_{k+1} = w_k - \alpha_k G^{-1}(w_k)\nabla F(w_k). \tag{6.24}$$

We can also replace $F(w)$ in (6.23) by a noisy first-order approximation, leading to a stochastic natural gradient iteration where $\nabla F(w_k)$ in (6.24) is replaced by a stochastic gradient estimate.

Both batch and stochastic versions of (6.24) resemble the quasi-Newton update rules discussed in §6.2. Instead of multiplying the gradient by the inverse of an approximation of the Hessian (which is not necessarily positive definite), it employs the positive semidefinite matrix $G(w_k)$ that expresses the local geometry of the space of prediction functions. In principle, this matrix does not even take into account the objective function F. However, as we shall now describe, one finds that these choices are all closely related in practice.

Practical Natural Gradient Because the discovery of the natural gradient algorithm is closely associated with information geometry, nearly all its applications involve density estimation [5, 33] or conditional probability estimation [119, 76, 99] using objective functions that are closely connected to the KL divergence. Natural gradient in this context is closely related to Fisher's scoring algorithm [118]. For instance, in the case of density estimation, the objective is usually the negative log likelihood

$$F(w) = \frac{1}{n}\sum_{i=1}^{n} -\log(h_w(x_i)) \approx \text{constant} + D_{KL}(P \| h_w),$$

where $\{x_1, \ldots, x_n\}$ represent independent training samples from an unknown distribution P. Recalling the expression of the Fisher information matrix (6.21) then clarifies its connection with the Hessian as one finds that

$$G(w) = -\mathbb{E}_{h_w}\left[\frac{\partial^2 \log(h_w(x))}{\partial w^2}\right] \quad \text{and} \quad \nabla^2 F(w) = -\mathbb{E}_P\left[\frac{\partial^2 \log(h_w(x))}{\partial w^2}\right].$$

These two expressions do not coincide in general because the expectations involve different distributions. However, when the natural gradient algorithm approaches the optimum, the parametric density h_{w_k} ideally approaches the true distribution P, in which case the Fisher information matrix $G(w_k)$ approaches the Hessian matrix $\nabla^2 F(w_k)$. This means that the natural gradient algorithm and Newton's method perform very similarly as optimality is approached.

Although it is occasionally possible to determine a convenient analytic expression [33, 76], the numerical computation of the Fisher information matrix $G(w_k)$ in large learning systems is generally very challenging. Moreover, estimating the expectation (6.21) with, say, a Monte-Carlo approach is usually prohibitive due to the cost of sampling the current density estimate h_{w_k}.

Several authors [119, 99] suggest to use instead a subset of training examples and compute a quantity of the form

$$\widetilde{G}(w_k) = \frac{1}{|S_k|} \sum_{i \in S_k} \left(\frac{\partial \log(h_w(x_i))}{\partial w}\bigg|_{w_k} \right) \left(\frac{\partial \log(h_w(x_i))}{\partial w}\bigg|_{w_k} \right)^T.$$

Although such algorithms are essentially equivalent to the generalized Gauss-Newton schemes described in §6.3, the natural gradient perspective comes with an interesting insight into the relation between the generalized Gauss-Newton matrix (6.17) and the Hessian matrix (6.5). Similar to the equality (6.21), these two matrices would be equal if the expectation was taken with respect to the model distribution h_w instead of the empirical sample distribution.

6.5 Methods that Employ Diagonal Scalings

The methods that we have discussed so far in this section are forced to overcome the fact that when employing an iteration involving an $\mathbb{R}^d \times \mathbb{R}^d$ scaling matrix, one needs to ensure that the improved per-iteration progress outweighs the added per-iteration cost. We have seen that these added costs can be as little as $4md$ operations and therefore amount to a moderate multiplicative factor on the cost of each iteration.

A strategy to further reduce this multiplicative factor, while still incorporating second-order-type information, is to restrict attention to *diagonal* or *block-diagonal* scaling matrices. Rather than perform a more general linear transformation through a symmetric positive definite matrix (i.e., corresponding to a scaling and rotation of the direction), the incorporation of a diagonal scaling matrix only has the effect of scaling the individual search direction components. This can be efficiently achieved by multiplying each coefficients of the gradient vector by the corresponding diagonal term of the scaling matrix, or, when the prediction function is linear, by adaptively renormalizing the input pattern coefficients [133].

Computing Diagonal Curvature A first family of algorithms that we consider directly computes the diagonal terms of the Hessian or Gauss-Newton matrix, then divides each coefficient of the stochastic gradient vector $g(w_k, \xi_k)$ by the corresponding diagonal term. Since the computation overhead of this operation is very small, it becomes important to make sure that the estimation of the diagonal terms of the curvature matrix is very efficient.

For instance, in the context of deep neural networks, [12] describes a back-propagation algorithm to efficiently compute the diagonal terms of the squared Jacobian matrix $J_h(w_k;\xi_k)^T J_h(w_k;\xi_k)$ that appears in the expression of the Gauss-Newton matrix (6.15). Each iteration of the proposed algorithm picks a training example, computes the stochastic gradient $g(w_k, \xi_k)$, updates a running estimate of the diagonal coefficients of the Gauss-Newton matrix by

$$\big[G_k\big]_i = (1-\lambda)\big[G_{k-1}\big]_i + \lambda\big[J_h(w_k;\xi_k)^T J_h(w_k;\xi_k)\big]_{ii} \quad \text{for some} \quad 0 < \lambda < 1,$$

then performs the scaled stochastic weight update

$$\big[w_{k+1}\big]_i = \big[w_k\big]_i - \left(\frac{\alpha}{\big[G_k\big]_i + \mu}\right)\big[g(w_k, \xi_k)\big]_i.$$

The small regularization constant $\mu > 0$ is introduced to handle situations where the Gauss-Newton matrix is singular or nearly singular. Since the computation of the diagonal of the squared Jacobian has a cost that is comparable to the cost of the computation of the stochastic gradient, each iteration of this algorithm is roughly twice as expensive as a first-order stochastic gradient iteration. The experience described in [12] shows that improvement in per-iteration progress can be sufficient to modestly outperform a well-tuned SG algorithm.

After describing this algorithm in later work [89, §9.1], the authors make two comments that illustrate well how this algorithm was used in practice. They first observe that the curvature only changes very slowly in the particular type of neural network that was considered. Due to this observation, a natural idea is to further reduce the computational overhead of the method by estimating the ratios $\alpha/([G_{k+1}]_i + \mu)$ only once every few epochs, for instance using a small subset of examples as in (6.15). The authors also mention that, as a rule of thumb, this diagonal scheme typically improves the convergence speed by only a factor of three relative to SG. Therefore, it might be more enlightening to view such an algorithm as a scheme to periodically retune a first-order SG approach rather than as a complete second-order method.

Estimating Diagonal Curvature Instead of explicitly computing the diagonal terms of the curvature matrix, one can follow the template of §6.2 and directly estimate the diagonal $[H_k]_i$ of the inverse Hessian using displacement pairs $\{(s_k, v_k)\}$ as defined in (6.13). For instance, [18] proposes to compute the scaling terms $[H_k]_i$ with the running average

$$\big[H_{k+1}\big]_i = (1-\lambda)\big[H_k\big]_i + \lambda \mathrm{Proj}\left(\frac{\big[s_k\big]_i}{\big[v_k\big]_i}\right),$$

where $\mathrm{Proj}(\cdot)$ represents a projection onto a predefined positive interval. It was later found that a direct application of (6.13) after a parameter update introduces a correlated noise that ruins the curvature estimate [19, §3]. Moreover, correcting this problem made the algorithm perform substantially worse because the chaotic behavior of the rescaling factors $[H_k]_i$ makes the choice of the stepsize α very difficult.

These problems can be addressed with a combination of two ideas [19, §5]. The first idea consists of returning to estimating the diagonal of the Hessian instead of the diagonal of this inverse, which amounts to working with the ratio $[v_k]_i/[s_k]_i$ instead of $[s_k]_i/[v_k]_i$. The second idea ensures that the effective stepsizes are monotonically decreasing by replacing the running average by the sum

$$\big[G_{k+1}\big]_i = \big[G_k\big]_i + \text{Proj}\left(\frac{\big[v_k\big]_i}{\big[s_k\big]_i}\right).$$

This effectively constructs a separate diminishing stepsize sequence $\alpha/[G_k]_i$ for each coefficient of the parameter vector. Keeping the curvature estimates in a fixed positive interval ensures that the effective stepsizes decrease at the rate $\mathcal{O}(1/k)$ as prescribed by Theorem 4.7, while taking the local curvature into account. This combination was shown to perform very well when the input pattern coefficients have very different variances [19], something that often happens, e.g., in text classification problems.

Diagonal Rescaling without Curvature The algorithms described above often require some form of regularization to handle situations where the Hessian matrix is (nearly) singular. To illustrate why this is needed, consider, e.g., optimization of the convex objective function

$$F(w_1, w_2) = \tfrac{1}{2}w_1^2 + \log(e^{w_2} + e^{-w_2}),$$

for which ones finds

$$\nabla F(w_1, w_2) = \begin{bmatrix} w_1 \\ \tanh(w_2) \end{bmatrix} \text{ and } \nabla^2 F(w_1, w_2) = \begin{bmatrix} 1 & 0 \\ 0 & 1/\cosh^2(w_2) \end{bmatrix}.$$

Performing a first-order gradient method update from a starting point of $(3, 3)$ yields the negative gradient step $-\nabla F \approx [-3, -1]$, which unfortunately does not point towards the optimum, namely the origin. Moreover, rescaling the step with the inverse Hessian actually gives a worse update direction $-(\nabla^2 F)^{-1}\nabla F \approx [-3, -101]$ whose large second component requires a small stepsize to keep the step well contained. Batch second-order optimization algorithms can avoid having to guess a good stepsize by using, e.g., line search techniques. Stochastic second-order algorithms, on the other hand, cannot rely on such procedures as easily.

This problem is of great concern in situations where the objective function is nonconvex. For instance, optimization algorithms for deep neural networks must navigate around saddle points and handle near-singular curvature matrices. It is therefore tempting to consider diagonal rescaling techniques that simply ensure equal progress along each axis, rather than attempt to approximate curvature very accurately.

For instance, RMSprop [152] estimates the average magnitude of each element of the stochastic gradient vector $g(w_k, \xi_k)$ by maintaining the running averages

$$\big[R_k\big]_i = (1 - \lambda)\big[R_{k-1}\big]_i + \lambda\big[g(w_k, \xi_k)\big]_i^2.$$

The rescaling operation then consists in dividing each component of $g(w_k, \xi_k)$ by the square root of the corresponding running average, ensuring that the expected second moment of each coefficient of the rescaled gradient is close to the unity:

$$\big[w_{k+1}\big]_i = \big[w_k\big]_i - \frac{\alpha}{\sqrt{\big[R_k\big]_i + \mu}}\big[g(w_k, \xi_k)\big]_i.$$

This surprisingly simple approach has shown to be very effective for the optimization of deep neural networks. Various improvement have been proposed [162, 84] on an empirical basis. The theorerical explanation of this performance on nonconvex problems is still the object of active research [43].

The popular ADAGRAD algorithm [54] can be viewed as a member of this family that replaces the running average by a sum:

$$\left[R_k\right]_i = \left[R_{k-1}\right]_i + \left[g(w_k, \xi_k)\right]_i^2.$$

In this manner, the approach constructs a sequence of diminishing effective stepsizes $\alpha/\sqrt{[R_k]_i + \mu}$ for each coefficient of the parameter vector. This algorithm was initially proposed and analyzed for the optimization of (not necessarily strongly) convex functions for which SG theory suggests diminishing stepsizes that scale with $\mathcal{O}(1/\sqrt{k})$. ADAGRAD is also known to perform well on deep learning networks, but one often finds that its stepsizes decrease too aggressively early in the optimization [162].

Structural Methods The performance of deep neural network training can of course be improved by employing better optimization algorithms. However, it can also be improved by changing the structure of the network in a manner that facilitates the optimization [80, 74]. We now describe one of these techniques, batch normalization [80], and discuss its relation to diagonal second-order methods.

Consider a particular fully connected layer in a deep neural network of the form discussed in §2.2. Using the notation of equation (2.4), the vector $x_i^{(j)}$ represents the input values of layer j when the network is processing the i-th training example. Omitting the layer index for simplicity, let $\hat{x}_i = (x_i^{(j)}, 1)$ denote the input vector augmented with an additional unit coefficient and let $\hat{w}_r = (W_{r1}, \ldots, W_{rd_{j-1}}, b_r)$ be the rth row of the matrix W_j augmented with the rth coefficient of the bias vector b_j. The layer outputs are then obtained by applying the activation function to the quantities $s_r = \hat{w}_r^T \hat{x}_i$ for $r \in \{1, \ldots, d_j\}$. Assuming for simplicity that all other parameters of the network are kept fixed, we can write

$$F(\hat{w}_1, \ldots, \hat{w}_{d_j}) = \frac{1}{n} \sum_{i=1}^{n} \ell(h(\hat{w}_1^T \hat{x}_i, \hat{w}_2^T \hat{x}_i, \ldots, \hat{w}_{d_j}^T \hat{x}_i), y_i),$$

where $h(s_1, \ldots, s_{d_j})$ encapsulates all subsequent layers in the network. The diagonal block of the Gauss-Newton matrix (6.16) corresponding to the parameters $\hat{w}_r$ then has the form

$$G_{[r]} = \frac{1}{|\mathcal{S}|} \sum_{i \in \mathcal{S}} \left[\left(\frac{dh}{ds_r} \right)^T \left(\frac{\partial^2 \ell}{\partial h^2} \right) \left(\frac{dh}{ds_r} \right) \right] \hat{x}_i \hat{x}_i^T, \tag{6.25}$$

which can be viewed as a weighted second moment matrix of the augmented input vectors $\{\hat{x}_i\}_{i \in \mathcal{S}}$. In particular, this matrix is perfectly conditioned if the weighted distribution of the layer inputs is white, i.e., they have zero mean and a unit covariance matrix. This could be achieved by first preprocessing the inputs by an affine transform that whitens their weighted distribution.

Two simplifications can drastically reduce the computational cost of this operation. First, we can ignore the bracketed coefficient in (6.25) and assume that we can use the same whitening transformation for all outputs $r \in \{1, \ldots, d_j\}$. Second, we can ignore the input cross-correlations and simply ensure that each input variable has zero mean and unit variance by replacing the input vector coefficients $\hat{x}_i[t]$ for each $t \in \{1, \ldots, d_{j-1}\}$ by the linearly transformed values $\alpha_t \hat{x}_i[t] + \beta_t$. Despite these simplifications, this normalization operation is very likely to improve the second order properties of the objective function. An important detail here is the computation of the normalization constants α_t and β_t. Estimating the mean and the standard deviation of each input with a simple running average works well if one expects these quantities to change very slowly. This is unfortunately not true in recent neural networks.[8]

Batch normalization [80] defines a special kind of neural network layer that performs this normalization using statistics collected with the current mini-batch of examples. The back-propagation algorithm that computes the gradients must of course be adjusted to account for the on-the-fly computation of the normalization coefficients. Assuming that one uses sufficiently large mini-batches, computing the statistics in this manner ensures that the normalization constants are very fresh. This comes at the price of making the output of the neural network on a particular training pattern dependent on the other patterns in the mini-batch. Since these other examples are *a priori* random, this amounts to generating additional noise in the stochastic gradient optimization. Although the variance of this noise is poorly controlled, inserting batch normalization layers in various points of a deep neural network is extremely effective and is now standard practice. Whether one can achieve the same improvement with more controlled techniques remains to be seen.

7 Other Popular Methods

Some optimization methods for machine learning are not well characterized as being within the two-dimensional schematic introduced in §3.4 (see Figure 3.3 on page 20), yet represent fundamentally unique approaches that offer theoretical and/or practical advantages. The purpose of this section is to discuss a few such ideas, namely, gradient methods with momentum, accelerated gradient methods, and coordinate descent methods. For ease of exposition, we introduce these techniques under the assumption that one is minimizing a continuously differentiable (not necessarily convex) function $F : \mathbb{R}^n \to \mathbb{R}$ and that full gradients can be computed in each iteration. Then, after each technique is introduced, we discuss how they may be applied in stochastic settings.

7.1 Gradient Methods with Momentum

Gradient methods with momentum are procedures in which each step is chosen as a combination of the steepest descent direction and the most recent iterate displacement. Specifically, with an initial point w_1, scalar sequences $\{\alpha_k\}$ and $\{\beta_k\}$ that are either predetermined or set dynamically, and $w_0 := w_1$, these methods are characterized by the iteration

$$w_{k+1} \leftarrow w_k - \alpha_k \nabla F(w_k) + \beta_k (w_k - w_{k-1}). \tag{7.1}$$

Here, the latter is referred to as the momentum term, which, recursively, maintains the algorithm's movement along previous search directions.

The iteration (7.1) can be motivated in various ways; e.g., it is named after the fact that it represents a discretization of a certain second-order ordinary differential equation with friction. Of

[8]This used to be true in the 1990s because neural networks were using bounded activation functions such as the sigmoid $s(x) = 1/(1 + e^{-x})$. However, many recent results were achieved using the ReLU activation function $s(x) = \max\{0, x\}$ which is unbounded and homogeneous. The statistics of the intermediate variables in such network can change extremely quickly during the first phases of the optimization process [86].

course, when $\beta_k = 0$ for all $k \in \mathbb{N}$, it reduces to the steepest descent method. When $\alpha_k = \alpha$ and $\beta_k = \beta$ for some constants $\alpha > 0$ and $\beta > 0$ for all $k \in \mathbb{N}$, it is referred to as the heavy ball method [123], which is known to yield a superior rate of convergence as compared to steepest descent with a fixed stepsize for certain functions of interest. For example, when F is a strictly convex quadratic with minimum and maximum eigenvalues given by $c > 0$ and $L \geq c$, respectively, steepest descent and the heavy ball method each yield a linear rate of convergence (in terms of the distance to the solution converging to zero) with contraction constants respectively given by

$$\frac{\kappa - 1}{\kappa + 1} \quad \text{and} \quad \frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \quad \text{where} \quad \kappa := \frac{L}{c} \geq 1. \tag{7.2}$$

Choosing (α, β) to achieve these rates requires knowledge of (c, L), which might be unavailable. Still, even without this knowledge, the heavy ball method often outperforms steepest descent.

Additional connections with (7.1) can be made when F is a strictly convex quadratic. In particular, if (α_k, β_k) is chosen optimally for all $k \in \mathbb{N}$, in the sense that the pair is chosen to solve

$$\min_{(\alpha,\beta)} \; F(w_k - \alpha \nabla F(w_k) + \beta(w_k - w_{k-1})), \tag{7.3}$$

then (7.1) is exactly the linear conjugate gradient (CG) algorithm. While the heavy ball method is a stationary iteration (in the sense that the pair (α, β) is fixed), the CG algorithm is nonstationary and its convergence behavior is relatively more complex; in particular, the step-by-step behavior of CG depends on the eigenvalue distribution of the Hessian of F [69]. That said, in contrast to the heavy ball method, CG has a finite convergence guarantee. This, along with the fact that problems with favorable eigenvalue distributions are quite prevalent, has lead to the great popularity of CG in a variety of situations. More generally, nonlinear CG methods, which also follow the procedure in (7.1), can be viewed as techniques that approximate the optimal values defined by (7.3) when F is not quadratic.

An alternative view of the heavy ball method is obtained by expanding (7.1):

$$w_{k+1} \leftarrow w_k - \alpha \sum_{j=1}^{k} \beta^{k-j} \nabla F(w_k);$$

thus, each step can be viewed as an exponenetially decaying average of past gradients. By writing the iteration this way, one can see that the steps tend to accumulate contributions in directions of persistent descent, while directions that oscillate tend to get cancelled, or at least remain small.

This latter interpretation provides some intuitive explanation as to why a stochastic heavy ball method, and stochastic gradient methods with momentum in general, might be successful in various settings. In particular, their practical performance have made them popular in the community working on training deep neural networks [150]. Replacing the true gradient with a stochastic gradient in (7.1), one obtains an iteration that, over the long run, tends to continue moving in directions that the stochastic gradients suggest are ones of improvement, whereas movement is limited along directions along which contributions of many stochastic gradients cancel each other out. Theoretical guarantees about the inclusion of momentum in stochastic settings are elusive, and although practical gains have been reported [92, 150], more experimentation is needed.

7.2 Accelerated Gradient Methods

A method with an iteration similar to (7.1), but with its own unique properties, is the accelerated gradient method proposed by Nesterov [107]. Written as a two-step procedure, it involves the updates

$$\begin{aligned} \tilde{w}_k &\leftarrow w_k + \beta_k(w_k - w_{k-1}) \\ \text{and} \quad w_{k+1} &\leftarrow \tilde{w}_k - \alpha_k \nabla F(\tilde{w}_k), \end{aligned} \tag{7.4}$$

which leads to the condensed form

$$w_{k+1} \leftarrow w_k - \alpha_k \nabla F(w_k + \beta_k(w_k - w_{k-1})) + \beta_k(w_k - w_{k-1}). \tag{7.5}$$

In this manner, it is easy to compare the approach with (7.1). In particular, one can describe their difference as being a reversal in the order of computation. In (7.1), one can imagine taking the steepest descent step and then applying the momentum term, whereas (7.5) results when one follows the momentum term first, then applies a steepest descent step (with the gradient evaluated at $\tilde{w}_k$, not at w_k).

While this difference may appear to be minor, it is well known that (7.5) with appropriately chosen $\alpha_k = \alpha > 0$ for all $k \in \mathbb{N}$ and $\{\beta_k\} \nearrow 1$ leads to an *optimal* iteration complexity when F is convex and continuously differentiable with a Lipschitz continuous gradient. Specifically, while in such cases a steepest descent method converges with a distance to the optimal value decaying with a rate $\mathcal{O}(\frac{1}{k})$, the iteration (7.5) converges with a rate $\mathcal{O}(\frac{1}{k^2})$, which is provably the best rate that can be achieved by a gradient method. Unfortunately, no intuitive explanation as to how Nesterov's method achieves this optimal rate has been widely accepted. Still, one cannot deny the analysis and the practical gains that the technique has offered.

Acceleration ideas have been applied in a variety of other contexts as well, including for the minimization of nonsmooth convex functions; see [96]. For now, we merely mention that when applied in stochastic settings—with stochastic gradients employed in place of full gradients—one can only hope that acceleration might improve the constants in the convergence rate offered in Theorem 4.7; i.e., the rate itself cannot be improved [1].

7.3 Coordinate Descent Methods

Coordinate descent (CD) methods are among the oldest in the optimization literature. As their name suggests, they operate by taking steps along coordinate directions: one attempts to minimize the objective with respect to a single variable while all others are kept fixed, then other variables are updated similarly in an iterative manner. Such a simple idea is easy to implement, so it is not surprising that CD methods have a long history in many settings. Their limitations have been documented and well understood for many years (more on these below), but one can argue that their advantages were not fully recognized until recent work in machine learning and statistics demonstrated their ability to take advantage of certain structures commonly arising in practice.

The CD method for minimizing $F : \mathbb{R}^d \to \mathbb{R}$ is given by the iteration

$$w_{k+1} \leftarrow w_k - \alpha_k \nabla_{i_k} F(w_k) e_{i_k}, \quad \text{where} \quad \nabla_{i_k} F(w_k) := \frac{\partial F}{\partial w^{i_k}}(w_k), \tag{7.6}$$

w^{i_k} represents the i_k-th element of the parameter vector, and e_{i_k} represents the i_k-th coordinate vector for some $i_k \in \{1, \ldots, d\}$. In other words, the solution estimates w_{k+1} and w_k differ only in their i_k-th element as a result of a move in the i_k-th coordinate from w_k.

Specific versions of the CD method are defined by the manner in which the sequences $\{\alpha_k\}$ and $\{i_k\}$ are chosen. In some applications, it is possible to choose α_k as the global minimizer of F from w_k along the i_k-th coordinate direction. An important example of this, which has contributed to the recent revival of CD methods, occurs when the objective function has the form $F(w) = q(w) + \|w\|_1$ when q is a convex quadratic. Here, the exact minimization along each coordinate is not only possible, but desirable as it promotes the generation of sparse iterates; see also §8. More often, an exact one-dimensional minimization of F is not practical, in which case one is typically satisfied with α_k yielding a sufficient reduction in F from w_k. For example, so-called second-order CD methods compute α_k as the minimizer of a quadratic model of F along the i_k-th coordinate direction.

Concerning the choice of i_k, one could select it in each iteration in at least three different ways: by cycling through $\{1, \ldots, d\}$; by cycling through a random reordering of these indices (with the indices reordered after each set of d steps); or simply by choosing an index randomly with replacement in each iteration. Randomized CD algorithms (represented by the latter two strategies) have superior theoretical properties than the cyclic method (represented by the first strategy) as they are less likely to choose an unfortunate series of coordinates; more on this below. However, it remains an open question whether such randomized algorithms are more effective in typical applications.

We mention in passing that it is also natural to consider a *block-coordinate descent* method in which a handful of elements are chosen in each iteration. This is is particularly effective when the objective function is (partially) block separable, which occurs in matrix factorization problems and least squares and logistic regression when each sample only depends on a few features. Clearly, in such settings, there are great advantages of a block-coordinate descent approach. However, since their basic properties are similar to the case of using a single index in each iteration, we focus on the iteration (7.6).

Convergence Properties Contrary to what intuition might suggest, a CD method is not guaranteed to converge when applied to minimize any given continuously differentiable function. Powell [127] gives an example of a *nonconvex* continuously differentiable function of three variables for which a cyclic CD method with α_k chosen by exact one-dimensional minimization cycles without converging to a solution, i.e., at any limit point the gradient of F is nonzero. Although one can argue that failures of this type are unlikely to occur in practice, particularly for a randomized CD method, they show the weakness of the myopic strategy in a CD method that considers only one variable at a time. This is in contrast with the full gradient method, which guarantees convergence to stationarity even when the objective is nonconvex.

On the other hand, if the objective F is strongly convex, the CD method will not fail and one can establish a linear rate of convergence. The analysis is very simple when using a constant stepsize and we present one such result to provide some insights into the tradeoffs that arise with a CD approach. Let us assume that ∇F is coordinate-wise Lipschitz continuous in the sense that, for all $w \in \mathbb{R}^d$, $i \in \{1, \ldots, d\}$, and $\Delta w^i \in \mathbb{R}$, there exists a constant $L_i > 0$ such that

$$|\nabla_i F(w + \Delta w^i e_i) - \nabla_i F(w)| \leq L_i |\Delta w^i|. \tag{7.7}$$

We then define the maximum coordinate-wise Lipschitz constant as

$$\widehat{L} := \max_{i \in \{1, \ldots, d\}} L_i.$$

Loosely speaking, $\widehat{L}$ is a bound on the curvature of the function along all coordinates.

Theorem 7.1. *Suppose that the objective function $F : \mathbb{R}^d \to \mathbb{R}$ is continuously differentiable, strongly convex with constant $c > 0$, and has a gradient that is coordinate-wise Lipschitz continuous with constants $\{L_1, \dots, L_d\}$. In addition, suppose that $\alpha_k = 1/\widehat{L}$ and i_k is chosen independently and uniformly from $\{1, \dots, d\}$ for all $k \in \mathbb{N}$. Then, for all $k \in \mathbb{N}$, the iteration (7.6) yields*

$$\mathbb{E}[F(w_{k+1})] - F_* \leq \left(1 - \frac{c}{d\widehat{L}}\right)^k (F(w_1) - F_*). \tag{7.8}$$

Proof. As Assumption 4.1 leads to (4.3), coordinate-wise Lipschitz continuity of ∇F yields

$$F(w_{k+1}) \leq F(w_k) + \nabla_{i_k} F(w_k)(w_{k+1}^{i_k} - w_k^{i_k}) + \tfrac{1}{2}\widehat{L}(w_{k+1}^{i_k} - w_k^{i_k})^2.$$

Thus, with the stepsize chosen as $\alpha_k = 1/\widehat{L}$, it follows that

$$F(w_{k+1}) - F(w_k) \leq -\tfrac{1}{\widehat{L}}\nabla_{i_k} F(w_k)^2 + \tfrac{1}{2\widehat{L}}\nabla_{i_k} F(w_k)^2 = -\tfrac{1}{2\widehat{L}}\nabla_{i_k} F(w_k)^2.$$

Taking expectations with respect to the distribution of i_k, one obtains

$$\begin{aligned}
\mathbb{E}_{i_k}[F(w_{k+1})] - F(w_k) &\leq -\tfrac{1}{2\widehat{L}}\mathbb{E}_{i_k}[\nabla_{i_k} F(w_k)^2] \\
&= -\tfrac{1}{2\widehat{L}}\left(\tfrac{1}{d}\sum_{i=1}^{d} \nabla_i F(w_k)^2\right) \\
&= -\tfrac{1}{2\widehat{L}d}\|\nabla F(w_k)\|_2^2.
\end{aligned}$$

Subtracting F_*, taking total expectations, recalling (4.12), and applying the above inequality repeatedly over the first $k \in \mathbb{N}$ iterations yields (7.8), as desired. □

A brief overview of the convergence properties of other CD methods under the assumptions of Theorem 7.1 and in other settings is also worthwhile. First, it is interesting to compare the result of Theorem 7.1 with a result obtained using the *deterministic* Gauss-Southwell rule, in which i_k is chosen in each iteration according to the largest (in absolute value) component of the gradient. Using this approach, one obtains a similar result in which c is replaced by $\hat{c}$, the strong convexity parameter as measured by the ℓ_1-norm [116]. Since $\frac{c}{n} \leq \hat{c} \leq c$, the Gauss-Southwell rule can be up to n times faster than a randomized strategy, but in the worst case it is no better (and yet more expensive due to the need to compute the full gradient vector in each iteration). Alternative methods have also been proposed in which, in each iteration, the index i_k is chosen randomly with probabilities proportional to L_i or according to the largest ratio $|\nabla_i F(w_k)|/\sqrt{L_i}$ [93, 110]. These strategies also lead to linear convergence rates with constants that are better in some cases.

Favorable Problem Structures Theorem 7.1 shows that a simple randomized CD method is linearly convergent with constant dependent on the parameter dimension d. At first glance, this appears to imply that such a method is less efficient than a standard full gradient method. However, in situations in which d coordinate updates can be performed at cost similar to the evaluation of one full gradient, the method is competitive with a full gradient method both theoretically and in practice. Classes of problems in this category include those in which the objective function is

$$F(w) = \frac{1}{n}\sum_{j=1}^{n} \tilde{F}_j(x_j^T w) + \sum_{i=1}^{d} \hat{F}_i(w^i), \tag{7.9}$$

where, for all $j \in \{1, \dots, n\}$, the function $\tilde{F}_j$ is continuously differentiable and dependent on the *sparse* data vector x_j, and, for all $i \in \{1, \dots, d\}$, the function $\hat{F}_i$ is a (nonsmooth) regularization function. Such a form arises in least squares and logistic regression; see also §8.

For example, consider an objective function of the form

$$f(w) = \frac{1}{2}\|Xw - y\|_2^2 + \sum_{i=1}^{d} \hat{F}_i(w^i) \quad \text{with} \quad X = \begin{bmatrix} x_1 & \cdots & x_n \end{bmatrix},$$

which might be the original function of interest or might represent a model of (7.9) in which the first term is approximated by a convex quadratic model. In this setting,

$$\nabla_{i_k} f(w_{k+1}) = x_{i_k}^T r_{k+1} + \hat{F}'_{i_k}(w_{k+1}^{i_k}) \quad \text{with} \quad r_{k+1} := Aw_{k+1} - b,$$

where, with $w_{k+1} = w_k + \beta_k e_{i_k}$, one may observe that $r_{k+1} = r_k + \beta_k x_{i_k}$. That is, since the residuals $\{r_k\}$ can be updated with cost proportional to the number of nonzeros in x_{i_k}, call it $\text{nnz}(x_{i_k})$, the overall cost of computing the search direction in iteration $k+1$ is also $\mathcal{O}(\text{nnz}(x_{i_k}))$. On the other hand, an evaluation of the entire gradient requires a cost of $\mathcal{O}(\sum_{i=1}^n \text{nnz}(x_i))$.

Overall, there exist various types of objective functions for which minimization by a CD method (with exact gradient computations) can be effective. These include objectives that are (partially) block separable (which arise in dictionary learning and non-negative matrix factorization problems), have structures that allow for the efficient computation of individual gradient components, or are diagonally dominant in the sense that each step along a coordinate direction yields a reduction in the objective proportional to that which would have been obtained by a step along the steepest descent direction. Additional settings in which CD methods are effective are online problems where gradient information with respect to a group of variables becomes available in time, in which case it is natural to update these variables as soon as information is received.

Stochastic Dual Coordinate Ascent What about *stochastic* CD methods? As an initial thought, one might consider the replacement of $\nabla_{i_k} F(w_k)$ in (7.6) with a stochastic approximation, but this is not typical since one can usually as easily compute a d-dimensional stochastic gradient to apply an SG method. However, an interesting setting for the application of stochastic CD methods arises when one considers approaches to minimize a convex objective function of the form (7.9) by maximizing its *dual*. In particular, defining the convex conjugate of $\tilde{F}_j$ as $\tilde{F}_j^\star(u) := \max_w (w^T u - \tilde{F}_j(w))$, the Fenchel-Rockafellar dual of (7.9) when $\hat{F}_i(\cdot) = \frac{\lambda}{2}(\cdot)^2$ for all $i \in \{1, \dots, d\}$ is given by

$$F_{dual}(v) = \frac{1}{n} \sum_{j=1}^{n} -\tilde{F}_j^\star(-v_j) - \frac{\lambda}{2} \left\| \frac{1}{\lambda n} \sum_{j=1}^{n} v_j x_j \right\|_2^2.$$

The stochastic dual coordinate ascent (SDCA) method [145] applied to a function of this form has an iteration similar to (7.6), except that negative gradient steps are replaced by gradient steps due to the fact that one aims to maximize the dual. At the conclusion of a run of the algorithm, the corresponding *primal* solution can be obtained as $w \leftarrow \frac{1}{\lambda n} \sum_{j=1}^n v_j x_j$. The per-iteration cost of this approach is on par with that of a stochastic gradient method.

Parallel CD Methods We close this section by noting that CD methods are also attractive when one considers the exploitation of parallel computation. For example, consider a multicore system in which the parameter vector w is stored in shared memory. Each core can then execute a CD iteration independently and in an asynchronous manner, where if d is large compared to the number of cores, then it is unlikely that two cores are attempting to update the same variable at the same time. Since, during the time it takes a core to perform an update, the parameter vector w has likely changed (due to updates produced by other cores), each update is being made based on slightly stale information. However, convergence of the method can be proved, and improves when one can bound the degree of staleness of each update. For further information and insight into these ideas, we refer the reader to [16, 98].

8 Methods for Regularized Models

Our discussion of structural risk minimization (see §2.3) highlighted the key role played by regularization functions in the formulation of optimization problems for machine learning. The optimization methods that we presented and analyzed in the subsequent sections (§3–§7) are all applicable when the objective function involves a smooth regularizer, such as the squared ℓ_2-norm. In this section, we expand our investigation by considering optimization methods that handle the regularization as a distinct entity, in particular when that function is *nonsmooth*. One such regularizer that deserves special attention is the ℓ_1-norm, which induces sparsity in the optimal solution vector. For machine learning, sparsity can be beneficial since it leads to simpler models, and hence can be seen as a form of *feature selection*, i.e., for biasing the optimization toward solutions where only a few elements of the parameter vector are nonzero.

Broadly, this section focuses on the nonsmooth optimization problem

$$\min_{w\in\mathbb{R}^d} \ \Phi(w) := F(w) + \lambda\Omega(w), \tag{8.1}$$

where $F : \mathbb{R}^d \to \mathbb{R}$ includes the composition of a loss and prediction function, $\lambda > 0$ is a regularization parameter, and $\Omega : \mathbb{R}^d \to \mathbb{R}$ is a convex, nonsmooth regularization function. Specifically, we pay special attention to methods for solving the problem

$$\min_{w\in\mathbb{R}^d} \ \phi(w) := F(w) + \lambda\|w\|_1. \tag{8.2}$$

As discussed in §2, it is often necessary to solve a series of such problems over a sequence of values for the parameter λ. For further details in terms of problem (8.2), we refer the reader to [147, 9] for examples in a variety of applications. However, in our presentation of optimization methods, we assume that λ has been prescribed and is fixed. We remark in passing that (8.2) has as a special case the well-known LASSO problem [151] when $F(w) = \|Aw - b\|_2^2$ for some $A \in \mathbb{R}^{n\times d}$ and $b \in \mathbb{R}^n$.

Although nondifferentiable, the regularized ℓ_1 problem (8.2) has a structure that can be exploited in the design of algorithms. The algorithms that have been proposed can be grouped into classes of first- or second-order methods, and distinguished as those that either minimize the nonsmooth objective directly, as in a *proximal gradient* method, or by approximately minimizing a sequence of more complicated models, such as in a *proximal Newton* method.

There exist other *sparsity-inducing* regularizers besides the ℓ_1-norm, including group-sparsity-inducing regularizers that combine ℓ_1- and ℓ_2-norms taken with respect to groups of variables

[147, 9], as well as the nuclear norm for optimization over spaces of matrices [32]. While we do not discuss other such regularizers in detail, our presentation of methods for ℓ_1-norm regularized problems represents how methods for alternative regularizers can be designed and characterized.

As in the previous sections, we introduce the algorithms in this section under the assumption that F is continuously differentiable and full, batch gradients can be computed for it in each iteration, commenting on stochastic method variants once motivating ideas have been described.

8.1 First-Order Methods for Generic Convex Regularizers

The fundamental algorithm in unconstrained smooth optimization is the gradient method. For solving problem (8.1), the *proximal gradient* method represents a similar fundamental approach. Given an iterate w_k, a generic proximal gradient iteration, with $\alpha_k > 0$, is given by

$$w_{k+1} \leftarrow \arg\min_{w\in\mathbb{R}^d} \left(F(w_k) + \nabla F(w_k)^T(w - w_k) + \frac{1}{2\alpha_k}\|w - w_k\|_2^2 + \lambda\Omega(w) \right). \tag{8.3}$$

The term *proximal* refers to the presence of the third term in the minimization problem on the right-hand side, which encourages the new iterate to be close to w_k. Notice that if the regularization (i.e., last) term were not present, then (8.3) exactly recovers the gradient method update $w_{k+1} \leftarrow w_k - \alpha_k \nabla F(w_k)$; hence, as previously, we refer to α_k as the stepsize parameter. On the other hand, if the regularization term is present, then, similar to the gradient method, each new iterate is found by minimizing a model formed by a first-order Taylor series expansion of the objective function plus a simple scaled quadratic. Overall, the only thing that distinguishes a proximal gradient method from the gradient method is the regularization term, which is left untouched and included explicitly in each step computation.

To show how an analysis similar to those seen in previous sections can be used to analyze (8.3), we prove the following theorem. In it, we show that if F is strongly convex and its gradient function is Lipschitz continuous, then the iteration yields a global linear rate of convergence to the optimal objective value provided that the stepsizes are sufficiently small.

Theorem 8.1. *Suppose that $F : \mathbb{R}^d \to \mathbb{R}$ is continuously differentiable, strongly convex with constant $c > 0$, and has a gradient that is Lipschitz continuous with constant $L > 0$. In addition, suppose that $\alpha_k = \alpha \in (0, 1/L]$ for all $k \in \mathbb{N}$. Then, for all $k \in \mathbb{N}$, the iteration (8.3) yields*

$$\Phi(w_{k+1}) - \Phi(w_*) \leq (1 - \alpha c)^k(\Phi(w_1) - \Phi(w_*)),$$

where $w_* \in \mathbb{R}^d$ is the unique global minimizer of Φ in (8.1).

Proof. Since $\alpha_k = \alpha \in (0, 1/L]$, it follows from (4.3) that

$$\begin{aligned}
\Phi(w_{k+1}) &= F(w_{k+1}) + \lambda\Omega(w_{k+1}) \\
&\leq F(w_k) + \nabla F(w_k)^T(w_{k+1} - w_k) + \tfrac{1}{2}L\|w_{k+1} - w_k\|_2^2 + \lambda\Omega(w_{k+1}) \\
&\leq F(w_k) + \nabla F(w_k)^T(w_{k+1} - w_k) + \tfrac{1}{2\alpha}\|w_{k+1} - w_k\|_2^2 + \lambda\Omega(w_{k+1}) \\
&\leq F(w_k) + \nabla F(w_k)^T(w - w_k) + \tfrac{1}{2\alpha}\|w - w_k\|_2^2 + \lambda\Omega(w) \quad \text{for all} \quad w \in \mathbb{R}^d,
\end{aligned}$$

where the last inequality follows since w_{k+1} is defined by (8.3). Representing $w = w_k + d$, we obtain

$$\begin{aligned}
\Phi(w_{k+1}) &\leq F(w_k) + \nabla F(w_k)^T d + \tfrac{1}{2\alpha}\|d\|_2^2 + \lambda\Omega(w_k + d) \\
&\leq F(w_k) + \nabla F(w_k)^T d + \tfrac{1}{2}c\|d\|_2^2 - \tfrac{1}{2}c\|d\|_2^2 + \tfrac{1}{2\alpha}\|d\|_2^2 + \lambda\Omega(w_k + d) \\
&\leq F(w_k + d) + \lambda\Omega(w_k + d) - \tfrac{1}{2}c\|d\|_2^2 + \tfrac{1}{2\alpha}\|d\|_2^2 \\
&= \Phi(w_k + d) + \tfrac{1}{2}(\tfrac{1}{\alpha} - c)\|d\|_2^2,
\end{aligned}$$

which for $d = -\alpha c(w_k - w_*)$ means that

$$\begin{aligned}
\Phi(w_{k+1}) &\leq \Phi(w_k - \alpha c(w_k - w_*)) + \tfrac{1}{2}(\tfrac{1}{\alpha} - c)\|\alpha c(w_k - w_*)\|_2^2 \\
&= \Phi(w_k - \alpha c(w_k - w_*)) + \tfrac{1}{2}\alpha c^2(1 - \alpha c)\|w_k - w_*\|_2^2.
\end{aligned} \tag{8.4}$$

On the other hand, since the c-strongly convex function Φ satisfies (e.g., see [108, pp. 63–64])

$$\begin{gathered}
\Phi(\tau w + (1-\tau)\overline{w}) \leq \tau\Phi(w) + (1-\tau)\Phi(\overline{w}) - \tfrac{1}{2}c\tau(1-\tau)\|w - \overline{w}\|_2^2 \\
\text{for all } (w, \overline{w}, \tau) \in \mathbb{R}^d \times \mathbb{R}^d \times [0,1],
\end{gathered} \tag{8.5}$$

we have (considering $\overline{w} = w_k$, $w = w_*$, and $\tau = \alpha c \in (0,1]$ in (8.5)) that

$$\begin{aligned}
\Phi(w_k - \alpha c(w_k - w_*)) &\leq \alpha c\Phi(w_*) + (1 - \alpha c)\Phi(w_k) - \tfrac{1}{2}c(\alpha c)(1 - \alpha c)\|w_k - w_*\|_2^2 \\
&= \alpha c\Phi(w_*) + (1 - \alpha c)\Phi(w_k) - \tfrac{1}{2}\alpha c^2(1 - \alpha c)\|w_k - w_*\|_2^2.
\end{aligned} \tag{8.6}$$

Combining (8.6) with (8.4) and subtracting $\Phi(w_*)$, it follows that

$$\Phi(w_{k+1}) - \Phi(w_*) \leq (1 - \alpha c)(\Phi(w_k) - \Phi(w_*)).$$

The result follows by applying this inequality repeated over the first $k \in \mathbb{N}$ iterations. $\square$

One finds in Theorem 8.1 an identical result as for a gradient method for minimizing a smooth strongly convex function. As in such methods, the choice of the stepsize α is critical in practice; the convergence guarantees demand that it be sufficiently small, but a value that is too small might unduly slow the optimization process.

The proximal gradient iteration (8.3) is practical only when the proximal mapping

$$\text{prox}_{\lambda\Omega,\alpha_k}(\tilde{w}) := \arg\min_{w \in \mathbb{R}^n} \left(\lambda\Omega(w) + \frac{1}{2\alpha_k}\|w - \tilde{w}\|_2^2\right)$$

can be computed efficiently. This can be seen in the fact that the iteration (8.3) can equivalently be written as $w_{k+1} \leftarrow \text{prox}_{\lambda\Omega,\alpha_k}(w_k - \alpha_k\nabla F(w_k))$, i.e., the iteration is equivalent to applying a proximal mapping to the result of a gradient descent step. Situations when the proximal mapping is inexpensive to compute include when Ω is the indicator function for a simple set, such as a polyhedral set, when it is the ℓ_1-norm, or, more generally, when it is separable.

A stochastic version of the proximal gradient method can be obtained, not surprisingly, by replacing $\nabla F(w_k)$ in (8.3) by a stochastic approximation $g(w_k, \xi_k)$. The iteration remains cheap to perform (since $F(w_k)$ can be ignored as it does not effect the computed step). The resulting method attains similar behavior as a stochastic gradient method; analyses can be found, e.g., in [140, 8].

We now turn our attention to the most popular nonsmooth regularizer, namely the one defined by the ℓ_1 norm.

8.1.1 Iterative Soft-Thresholding Algorithm (ISTA)

In the context of solving the ℓ_1-norm regularized problem (8.2), the proximal gradient method is

$$w_{k+1} \leftarrow \arg\min_{w\in\mathbb{R}^d} \left(F(w_k) + \nabla F(w_k)^T(w - w_k) + \frac{1}{2\alpha_k}\|w - w_k\|_2^2 + \lambda\|w\|_1 \right). \tag{8.7}$$

The optimization problem on the right-hand side of this expression is separable and can be solved in closed form. The solution can be written component-wise as

$$[w_{k+1}]_i \leftarrow \begin{cases} [w_k - \alpha_k\nabla F(w_k)]_i + \alpha_k\lambda & \text{if } [w_k - \alpha_k\nabla F(w_k)]_i < -\alpha_k\lambda \\ 0 & \text{if } [w_k - \alpha_k\nabla F(w_k)]_i \in [-\alpha_k\lambda, \alpha_k\lambda] \\ [w_k - \alpha_k\nabla F(w_k)]_i - \alpha_k\lambda & \text{if } [w_k - \alpha_k\nabla F(w_k)]_i > \alpha_k\lambda. \end{cases} \tag{8.8}$$

One also finds that this iteration can be written, with $(\cdot)_+ := \max\{\cdot, 0\}$, as

$$w_{k+1} \leftarrow \mathcal{T}_{\alpha_k\lambda}(w_k - \alpha_k\nabla F(w_k)), \quad \text{where} \quad [\mathcal{T}_{\alpha_k\lambda}(\tilde{w})]_i = (|\tilde{w}_i| - \alpha_k\lambda)_+\operatorname{sgn}(\tilde{w}_i). \tag{8.9}$$

In this form, $\mathcal{T}_{\alpha_k\lambda}$ is referred to as the soft-thresholding operator, which leads to the name *iterative soft-thresholding algorithm* (ISTA) being used for (8.7)–(8.8) [53, 42].

It is clear from (8.8) that the ISTA iteration induces sparsity in the iterates. If the steepest descent step with respect to F yields a component with absolute value less than $\alpha_k\lambda$, then that component is set to zero in the subsequent iterate; otherwise, the operator still has the effect of shrinking components of the solution estimates in terms of their magnitudes. When only a stochastic estimate $g(w_k, \xi_k)$ of the gradient is available, it can be used instead of $\nabla F(w_k)$.

A variant of ISTA with acceleration (recall §7.2), known as FISTA [10], is popular in practice. We also mention that effective techniques have been developed for computing the stepsize α_k, in ISTA or FISTA, based on an estimate of the Lipschitz constant of ∇F or on curvature measured in recent iterations [10, 159, 11].

8.1.2 Bound-Constrained Methods for ℓ_1-norm Regularized Problems

By observing the structure created by the ℓ_1-norm, one finds that an equivalent *smooth* reformulation of problem (8.2) is easily derived. In particular, by writing $w = u - v$ where u plays the *positive part* of w while v plays the *negative part*, problem (8.2) can equivalently be written as

$$\min_{(u,v)\in\mathbb{R}^d\times\mathbb{R}^d} \tilde{\phi}(u, v) \quad \text{s.t.} \quad (u, v) \geq 0, \quad \text{where} \quad \tilde{\phi}(u, v) = F(u - v) + \lambda\sum_{i=1}^d (u_i + v_i). \tag{8.10}$$

Now with a bound-constrained problem in hand, one has at their disposal a variety of optimization methods that have been developed in the optimization literature.

The fundamental iteration for solving bound-constrained optimization problems is the *gradient projection* method. In the context of (8.10), the iteration reduces to

$$\begin{bmatrix} u_{k+1} \\ v_{k+1} \end{bmatrix} \leftarrow P_+\left(\begin{bmatrix} u_k \\ v_k \end{bmatrix} - \alpha_k \begin{bmatrix} \nabla_u\tilde{\phi}(u_k, v_k) \\ \nabla_v\tilde{\phi}(u_k, v_k) \end{bmatrix} \right) = P_+\left(\begin{bmatrix} u_k - \alpha_k\nabla F(u_k - v_k) - \alpha_k\lambda e \\ v_k + \alpha_k\nabla F(u_k - v_k) - \alpha_k\lambda e \end{bmatrix} \right), \tag{8.11}$$

where P_+ projects onto the nonnegative orthant and $e \in \mathbb{R}^d$ is a vector of ones.

Interestingly, the gradient projection method can also be derived from the perspective of a proximal gradient method where the regularization term Ω is chosen to be the indicator function for the feasible set (a box). In this case, the mapping $\mathcal{T}_{\alpha_k\lambda}$ is replaced by the projection operator onto the bound constraints, causing the corresponding proximal gradient method to coincide with the gradient projection method. In the light of this observation, one should expect the iteration (8.11) to inherit the property of being globally linearly convergent when F satisfies the assumptions of Theorem 8.1. However, since the variables in (8.10) have been split into positive and negative parts, this property is maintained *only if* the iteration maintains complementarity of each iterate pair, i.e., if $[u_k]_i[v_k]_i = 0$ for all $k \in \mathbb{N}$ and $i \in \{1, \ldots, d\}$. This behavior is also critical for the practical performance of the method in general since, without it, the algorithm would not generate sparse solutions. In particular, maintaining this property allows the algorithm to be implemented in such a way that one effectively only needs d optimization variables.

A natural question that arises is whether the iteration (8.11) actually differs from an ISTA iteration, especially given that both are built upon proximal gradient ideas. In fact, the iterations can lead to the same update, but do not always. Consider, for example, an iterate $w_k = u_k - v_k$ such that for $i \in \{1, \ldots, d\}$ one finds $[w_k]_i > 0$ with $[u_k]_i = [w_k]_i$ and $[v_k]_i = 0$. (A similar look, with various signs reversed, can be taken when $[w_k]_i < 0$.) If $[w_k - \alpha_k \nabla F(w_k)]_i > \alpha_k\lambda$, then (8.8) and (8.11) yield

$$\begin{aligned} [w_{k+1}]_i &\leftarrow [w_k - \alpha_k \nabla F(w_k)]_i - \alpha_k\lambda > 0 \\ \text{and} \quad [u_{k+1}]_i &\leftarrow [u_k - \alpha_k \nabla F(w_k)]_i - \alpha_k\lambda > 0. \end{aligned}$$

However, it is important to note the step taken in the negative part; in particular, if $[\nabla F(w_k)]_i \leq \lambda$, then $[v_{k+1}]_i \leftarrow 0$, but, if $[\nabla F(w_k)]_i > \lambda$, then $[v_{k+1}]_i \leftarrow \alpha_k \nabla F(w_k) - \alpha_k\lambda$, in which case the lack of complementarity between u_{k+1} and v_{k+1} should be rectified. A more significant difference arises when, e.g., $[w_k - \alpha_k \nabla F(w_k)]_i < -\alpha_k\lambda$, in which case (8.8) and (8.11) yield

$$\begin{aligned} [w_{k+1}]_i &\leftarrow [w_k - \alpha_k \nabla F(w_k)]_i + \alpha_k\lambda < 0, \\ [u_{k+1}]_i &\leftarrow 0, \\ \text{and} \quad [v_{k+1}]_i &\leftarrow [v_k + \alpha_k \nabla F(w_k)]_i - \alpha_k\lambda > 0. \end{aligned}$$

The pair $([u_{k+1}]_i, [v_{k+1}]_i)$ are complementary, but $[w_{k+1}]_i$ and $[-v_{k+1}]_i$ differ by $[w_k]_i > 0$.

Several first-order [59, 58] and second-order [138] gradient projection methods have been proposed to solve (8.2). Such algorithms should be preferred over similar techniques for general bound-constrained optimization, e.g., those in [163, 95], since such general techniques may be less effective by not exploiting the structure of the reformulation (8.10) of (8.2).

A stochastic projected gradient method, with $\nabla F(w_k)$ replaced by $g(w_k, \xi_k)$, has similar convergence properties as a standard SG method; e.g., see [105]. These properties apply in the present context, but also apply when a proximal gradient method is used to solve (8.1) when Ω represents the indicator function of a box constraint.

8.2 Second-Order Methods

We now turn our attention to methods that, like Newton's method for smooth optimization, are designed to solve regularized problems through successive minimization of second-order models constructed along the iterate sequence $\{w_k\}$. As in a proximal gradient method, the smooth function F is approximated by a Taylor series and the regularization term is kept unchanged. We focus on two classes of methods for solving (8.2): *proximal Newton* and *orthant-based* methods.

Both classes of methods fall under the category of *active-set* methods. One could also consider the application of an *interior-point* method to solve the bound-constrained problem (8.10) [114]. This, by its nature, constitutes a second-order method that would employ Hessians of F or corresponding quasi-Newton approximations. However, a disadvantage of the interior-point approach is that, by staying away from the boundary of the feasible region, it does not promote the fast generation of sparse solutions, which is in stark contrast with the methods described below.

8.2.1 Proximal Newton Methods

We use the term *proximal Newton* to refer to techniques that directly minimize the nonsmooth function arising as the sum of a quadratic model of F and the regularizer. In particular, for solving problem (8.2), a proximal Newton method is one that constructs, at each $k \in \mathbb{N}$, a model

$$q_k(w) = F(w_k) + \nabla F(w_k)^T(w - w_k) + \frac{1}{2}(w - w_k)^T H_k(w - w_k) + \lambda\|w\|_1 \approx \phi(w). \tag{8.12}$$

where H_k represents $\nabla^2 F(w_k)$ or a quasi-Newton approximation of it. This model has a similar form as the one in (8.7), except that the simple quadratic is replaced by the quadratic form defined by H_k. A proximal Newton method would involve (approximately) minimizing this model to compute a trial iterate $\tilde{w}_k$, then a stepsize $\alpha_k > 0$ would be taken from a predetermined sequence or chosen by a line search to ensure that the new iterate $w_{k+1} \leftarrow w_k + \alpha_k(\tilde{w}_k - w_k)$ yields $\Phi(w_{k+1}) < \Phi(w_k)$.

Proximal Newton methods are more challenging to design, analyze, and implement than proximal gradient methods. That being said, they can perform better in practice once a few key challenges are addressed. The three ingredients below have proved to be essential in ensuring the practical success and scalability of a proximal Newton method. For simplicity, we assume throughout that H_k has been chosen to be positive definite.

Choice of Subproblem Solver The model q_k inherits the nonsmooth structure of ϕ, which has the benefit of allowing a proximal Newton method to cross manifolds of nondifferentiability while simultaneously promoting sparsity of the iterates. However, the method needs to overcome the fact that the model q_k is nonsmooth, which makes the subproblem for minimizing q_k challenging. Fortunately, with the particular structure created by a quadratic plus an ℓ_1-norm term, various methods are available for minimizing such a nonsmooth function. For example, coordinate descent is particularly well-suited in this context [153, 78] since the global minimizer of q_k along a coordinate descent direction can be computed analytically. Such a minimizer often occurs at a point of nondifferentiability (namely, when a component is zero), thus ensuring that the method will generate sparse iterates. Updating the gradient of the model q_k after each coordinate descent step can also be performed efficiently, even if H_k is given as a limited memory quasi-Newton approximation [137]. Numerical experiments have shown that employing a coordinate descent iteration is more efficient in certain applications than employing, say, an ISTA iteration to minimize q_k, though the latter is also a viable strategy in some applications.

Inaccurate Subproblem Solves A proximal Newton method needs to overcome the fact that, in large-scale settings, it is impractical to minimize q_k accurately for all $k \in \mathbb{N}$. Hence, it is natural to consider the computation of an inexact minimizer of q_k. The issue then becomes: what are practical, yet theoretically sufficient termination criteria when computing an approximate minimizer of the nonsmooth function q_k? A common suggestion in the literature has been to use the norm of the minimum-norm subgradient of q_k at a given approximate minimizer. However, this measure is not continuous, making it inadequate for these purposes.[9] Interestingly, the norm of an ISTA step is an appropriate measure. In particular, letting $\text{ista}_k(w)$ represent the result of an ISTA step applied to q_k from w, the value $\|\text{ista}_k(w) - w\|_2$ satisfies the following two key properties: (i) it equals zero if and only if w is a minimizer of q_k, and (ii) it varies continuously over $\mathbb{R}^d$.

Complete and sufficient termination criteria are then as follows: a trial point $\tilde{w}_k$ represents a sufficiently accurate minimizer of q_k if, for some $\eta \in [0, 1)$, one finds

$$\|\text{ista}_k(\tilde{w}_k) - \tilde{w}_k\|_2 \leq \eta \|\text{ista}(w_k) - w_k\|_2 \quad \text{and} \quad q_k(\tilde{w}_k) < q_k(w_k).$$

The latter condition, requiring a decrease in q_k, must also be imposed since the ISTA criterion alone does not exert sufficient control to ensure convergence. Employing such criteria, it has been observed to be efficient to perform the minimization of q_k inaccurately at the start, and to increase the accuracy of the model minimizers as one approaches the solution. A superlinear rate of convergence for the overall proximal Newton iteration can be obtained by replacing η by η_k where $\{\eta_k\} \searrow 0$, along with imposing a stronger descent condition on the decrease in q_k [31].

Elimination of Variables Due to the structure created by the ℓ_1-norm regularizer, it can be effective in some applications to first identify a set of *active* variables—i.e., ones that are predicted to be equal to zero at a minimizer for q_k—then compute an approximate minimizer of q_k over the remaining *free* variables. Specifically, supposing that a set $\mathcal{A}_k \subseteq \{1, \dots, d\}$ of active variables has been identified, one may compute an (approximate) minimizer of q_k by (approximately) solving the reduced-space problem

$$\min_{w \in \mathbb{R}^d} \; q_k(w) \quad \text{s.t.} \quad [w]_i = 0, \; i \in \mathcal{A}_k. \tag{8.13}$$

Moreover, during the minimization process for this problem, one may have reason to believe that the process may be improved by adding or removing elements from the active set estimate $\mathcal{A}_k$. In any case, performing the elimination of variables imposed in (8.13) has the effect of reducing the size of the subproblem, and can often lead to fewer iterations being required in the overall proximal Newton method. How should the active set $\mathcal{A}_k$ be defined? A technique that has become popular recently is to use sensitivity information as discussed in more detail in the next subsection.

8.2.2 Orthant-Based Methods

Our second class of second-order methods is based on the observation that the ℓ_1-norm regularized objective ϕ in problem (8.2) is smooth in any orthant in $\mathbb{R}^d$. Based on this observation, orthant-based methods construct, in every iteration, a smooth quadratic model of the objective, then produce a search direction by minimizing this model. After performing a line search designed to reduce the objective function, a new orthant is selected and the process is repeated. This approach can be motivated by the success of second-order gradient projection methods for bound constrained optimization, which at every iteration employ a gradient projection search to identify an active set and perform a minimization of the objective function over a space of free variables to compute a search direction.

[9] Consider the one-dimensional case of having $q_k(w) = |w|$. The minimum-norm subgradient of q_k has a magnitude of 1 at all points, except at the minimizer $w^* = 0$; hence, this norm does not provide a measure of proximity to w^*.

The selection of an orthant is typically done using sensitivity information. Specifically, with the minimum norm subgradient of ϕ at $w \in \mathbb{R}^d$, which is given component-wise for all $i \in \{1, \dots, d\}$ by

$$\hat{g}_i(w) = \begin{cases} [\nabla F(w)]_i + \lambda & \text{if } w_i > 0 \text{ or } \{w_i = 0 \text{ and } [\nabla F(w)]_i + \lambda < 0\} \\ [\nabla F(w)]_i - \lambda & \text{if } w_i < 0 \text{ or } \{w_i = 0 \text{ and } [\nabla F(w)]_i - \lambda > 0\} \\ 0 & \text{otherwise,} \end{cases} \tag{8.14}$$

the active orthant for an iterate w_k is characterized by the sign vector

$$\zeta_{k,i} = \begin{cases} \text{sgn}([w_k]_i) & \text{if } [w_k]_i \neq 0 \\ \text{sgn}(-[\hat{g}(w_k)]_i) & \text{if } [w_k]_i = 0. \end{cases} \tag{8.15}$$

Along these lines, let us also define the subsets of $\{1, \dots, d\}$ given by

$$\mathcal{A}_k = \{i : [w_k]_i = 0 \text{ and } |[\nabla F(w_k)]_i| \leq \lambda\} \tag{8.16}$$

$$\text{and } \mathcal{F}_k = \{i : [w_k]_i \neq 0\} \cup \{i : [w_k]_i = 0 \text{ and } |[\nabla F(w_k)]_i| > \lambda\}, \tag{8.17}$$

where $\mathcal{A}_k$ represents the indices of variables that are active and kept at zero while $\mathcal{F}_k$ represents those that are free to move.

Given these quantities, an orthant-based method proceeds as follows. First, one computes the (approximate) solution d_k of the (smooth) quadratic problem

$$\begin{aligned} \min_{d \in \mathbb{R}^n} \ & \hat{g}(w_k)^T d + \tfrac{1}{2} d^T H_k d \\ \text{s.t. } & d_i = 0, \quad i \in \mathcal{A}_k, \end{aligned}$$

where H_k represents $\nabla^2 F(x^k)$ or an approximation of it. Then, the algorithm performs a line search—over a path contained in the current orthant—to compute the next iterate. For example, one option is a projected backtracking line search along d_k, which involves computing the largest α_k in a decreasing geometric sequence so

$$F(P_k(w_k + \alpha_k d_k)) < F(w_k).$$

Here, $P_k(w)$ projects $w \in \mathbb{R}^d$ onto the orthant defined by ζ^k, i.e.,

$$[P_k(w)]_i = \begin{cases} w_i & \text{if } \text{sgn}(w_i) = \zeta_{k,i} \\ 0 & \text{otherwise.} \end{cases} \tag{8.18}$$

In this way, the initial and final points of an iteration lie in the same orthant. Orthant-based methods have proved to be quite useful in practice; e.g., see [7, 30].

Commentary Proximal Newton and orthant-based methods represent two efficient classes of second-order active-set methods for solving the ℓ_1-norm regularized problem (8.2). The proximal Newton method is reminiscent of the sequential quadratic programming method (SQP) for constrained optimization; they both solve a complex subproblem that yields a useful estimate of the optimal active set. Although solving the piecewise quadratic model (8.12) is very expensive in general, the coordinate descent method has proven to be well suited for this task, and allows the proximal Newton method to be applied to very large problems [79]. Orthant-based methods have shown to be equally useful, but in a more heuristic way, since some popular implementations lack convergence guarantees [58, 30]. Stochastic variants of both proximal Newton and orthant-based schemes can be devised in natural ways, and generally inherit the properties of stochastic proximal gradient methods as long as the Hessian approximations are forced to possess eigenvalues within a positive interval.

9 Summary and Perspectives

Mathematical optimization is one of the foundations of machine learning, touching almost every aspect of the discipline. In particular, numerical optimization algorithms, the main subject of this paper, have played an integral role in the transformational progress that machine learning has experienced over the past two decades. In our study, we highlight the dominant role played by the stochastic gradient method (SG) of Robbins and Monro [130], whose success derives from its superior work complexity guarantees. A concise, yet broadly applicable convergence and complexity theory for SG is presented here, providing insight into how these guarantees have translated into practical gains.

Although the title of this paper suggests that our treatment of optimization methods for machine learning is comprehensive, much more could be said about this rapidly evolving field. Perhaps most importantly, we have neither discussed nor analyzed at length the opportunities offered by parallel and distributed computing, which may alter our perspectives in the years to come. In fact, it has already been widely acknowledged that SG, despite its other benefits, may not be the best suited method for emerging computer architectures.

This leads to our discussion of a spectrum of methods that have the potential to surpass SG in the next generation of optimization methods for machine learning. These methods, such as those built on noise reduction and second-order techniques, offer the ability to attain improved convergence rates, overcome the adverse effects of high nonlinearity and ill-conditioning, and exploit parallelism and distributed architectures in new ways. There are important methods that are not included in our presentation—such as the alternating direction method of multipliers (ADMM) [57, 64, 67] and the expectation-maximization (EM) method and its variants [48, 160]—but our study covers many of the core algorithmic frameworks in optimization for machine learning, with emphasis on methods and theoretical guarantees that have the largest impact on practical performance.

With the great strides that have been made and the various avenues for continued contributions, numerical optimization promises to continue to have a profound impact on the rapidly growing field of machine learning.

A Convexity and Analyses of SG

Our analyses of SG in §4 can be characterized as relying primarily on smoothness in the sense of Assumption 4.1. This has advantages and disadvantages. On the positive side, it allows us to prove convergence results that apply equally for the minimization of convex and nonconvex functions, the latter of which has been rising in importance in machine learning; recall §2.2. It also allows us to present results that apply equally to situations in which the stochastic vectors are unbiased estimators of the gradient of the objective, or when such estimators are scaled by a symmetric positive definite matrix; recall (4.2). A downside, however, is that it requires us to handle the minimization of nonsmooth models separately, which we do in §8.

As an alternative, a common tactic employed by many authors is to leverage convexity instead of smoothness, allowing for the establishment of guarantees that can be applied in smooth and nonsmooth settings. For example, a typical approach for analyzing stochastic gradient-based methods is to commence with the following fundamental equations related to squared distances to the optimum:

$$\begin{aligned}\|w_{k+1} - w_*\|_2^2 - \|w_k - w_*\|_2^2 &= 2(w_{k+1} - w_k)^T(w_k - w_*) + \|w_{k+1} - w_k\|_2^2 \\ &= -2\alpha_k g(w_k, \xi_k)^T(w_k - w_*) + \alpha_k^2\|g(w_k, \xi_k)\|_2^2. \end{aligned} \tag{A.1}$$

Assuming that $\mathbb{E}_{\xi_k}[g(w_k, \xi_k)] = \hat{g}(w_k) \in \partial F(w_k)$, one then obtains

$$\begin{aligned} &\mathbb{E}_{\xi_k}[\|w_{k+1} - w_*\|_2^2] - \|w_k - w_*\|_2^2 \\ = &\; -2\alpha_k \hat{g}(w_k)^T(w_k - w_*) + \alpha_k^2\mathbb{E}_{\xi_k}[\|g(w_k, \xi_k)\|_2^2], \end{aligned} \tag{A.2}$$

which has certain similarities with (4.10a). One can now introduce an assumption of convexity to bound the first term on the right-hand side in this expression; in particular, convexity offers the subgradient inequality

$$\hat{g}(w_k)^T(w_k - w_*) \geq F(w_k) - F(w_*) \geq 0$$

while strong convexity offers the stronger condition (4.11). Combined with a suitable assumption on the second moment of the stochastic subgradients to bound the second term in the expression, the entire right-hand side can be adequately controlled through judicious stepsize choices. The resulting analysis then has many similarities with that presented in §4, especially if one introduces an assumption about Lipscithz continuity of the gradients of F in order to translate results on decreases in the distance to the solution in terms of decreases in F itself. The interested reader will find a clear exposition of such results in [105].

Note, however, that one can see in (A.2) that analyses based on distances to the solution do not carry over easily to nonconvex settings or when (quasi-)Newton type steps are employed. In such situations, without explicit knowledge of w_*, one cannot easily ensure that the first term on the right-hand side can be bounded appropriately.

B Proofs

Inequality (4.3). Under Assumption 4.1, one obtains

$$\begin{aligned} F(w) &= F(\overline{w}) + \int_0^1 \frac{\partial F(\overline{w} + t(w - \overline{w}))}{\partial t}\, dt \\ &= F(\overline{w}) + \int_0^1 \nabla F(\overline{w} + t(w - \overline{w}))^T(w - \overline{w})\, dt \\ &= F(\overline{w}) + \nabla F(\overline{w})^T(w - \overline{w}) + \int_0^1 \left[\nabla F(\overline{w} + t(w - \overline{w})) - \nabla F(\overline{w})\right]^T(w - \overline{w})\, dt \\ &\leq F(\overline{w}) + \nabla F(\overline{w})^T(w - \overline{w}) + \int_0^1 L\|t(w - \overline{w})\|_2\|w - \overline{w}\|_2\, dt, \end{aligned}$$

from which the desired result follows. □

Inequality (4.12). Given $w \in \mathbb{R}^d$, the quadratic model

$$q(\overline{w}) := F(w) + \nabla F(w)^T(\overline{w} - w) + \tfrac{1}{2}c\|\overline{w} - w\|_2^2$$

has the unique minimizer $\overline{w}_* := w - \frac{1}{c}\nabla F(w)$ with $q(\overline{w}_*) = F(w) - \frac{1}{2c}\|\nabla F(w)\|_2^2$. Hence, the inequality (4.11) with $\overline{w} = w_*$ and any $w \in \mathbb{R}^d$ yields

$$F_* \geq F(w) + \nabla F(w)^T(w_* - w) + \tfrac{1}{2}c\|w_* - w\|_2^2 \geq F(w) - \tfrac{1}{2c}\|\nabla F(w)\|_2^2,$$

from which the desired result follows. □

Corollary 4.12. Define $G(w) := \|\nabla F(w)\|_2^2$ and let L_G be the Lipschitz constant of $\nabla G(w) = 2\nabla^2 F(w)\nabla F(w)$. Then,

$$\begin{aligned} G(w_{k+1}) - G(w_k) &\leq \nabla G(w_k)^T(w_{k+1} - w_k) + \tfrac{1}{2}L_G\|w_{k+1} - w_k\|_2^2 \\ &\leq -\alpha_k \nabla G(w_k)^T g(w_k, \xi_k) + \tfrac{1}{2}\alpha_k^2 L_G\|g(w_k, \xi_k)\|_2^2. \end{aligned}$$

Taking the expectation with respect to the distribution of ξ_k, one obtains from Assumptions 4.1 and 4.3 and the inequality (4.9) that

$$\begin{aligned} &\mathbb{E}_{\xi_k}[G(w_{k+1}] - G(w_k) \\ &\leq \ -2\alpha_k \nabla F(w_k)^T \nabla^2 F(w_k)^T E_{\xi_k}[g(w_k, \xi_k)] + \tfrac{1}{2}\alpha_k^2 L_G E_{\xi_k}[\|g(w_k, \xi_k)\|_2^2] \\ &\leq 2\alpha_k \|\nabla F(w_k)\|_2 \|\nabla^2 F(w_k)\|_2 \|E_{\xi_k}[g(w_k, \xi_k)]\|_2 + \tfrac{1}{2}\alpha_k^2 L_G E_{\xi_k}[\|g(w_k, \xi_k)\|_2^2] \\ &\leq 2\alpha_k L\mu_G \|\nabla F(w_k)\|_2^2 + \tfrac{1}{2}\alpha_k^2 L_G(M + M_V\|\nabla F(w_k)\|_2^2). \end{aligned}$$

Taking the total expectation simply yields

$$\begin{aligned} &\mathbb{E}[G(w_{k+1})] - \mathbb{E}[G(w_k)] \\ &\leq 2\alpha_k L\mu_G \mathbb{E}[\|\nabla F(w_k)\|_2^2] + \tfrac{1}{2}\alpha_k^2 L_G(M + M_V \mathbb{E}[\|\nabla F(w_k)\|_2^2]). \end{aligned} \tag{B.1}$$

Recall that Theorem 4.10 establishes that the first component of this bound is the term of a convergent sum. The second component of this bound is also the term of a convergent sum since $\sum_{k=1}^{\infty} \alpha_k^2$ converges and since $\alpha_k^2 \leq \alpha_k$ for sufficiently large $k \in \mathbb{N}$, meaning that again the result of Theorem 4.10 can be applied. Therefore, the right-hand side of (B.1) is the term of a convergent sum. Let us now define

$$\begin{aligned} S_K^+ &= \sum_{k=1}^{K} \max(0, \mathbb{E}[G(w_{k+1})] - \mathbb{E}[G(w_k)]) \\ \text{and} \quad S_K^- &= \sum_{k=1}^{K} \max(0, \mathbb{E}[G(w_k)] - \mathbb{E}[G(w_{k+1})]). \end{aligned}$$

Since the bound (B.1) is positive and forms a convergent sum, the nondecreasing sequence S_K^+ is upper bounded and therefore converges. Since, for any $K \in \mathbb{N}$, one has $G(w_K) = G(w_0) + S_K^+ - S_K^- \geq 0$, the nondecreasing sequence S_K^- is upper bounded and therefore also converges. Therefore $G(w_K)$ converges and Theorem 4.9 means that this limit must be zero. □

References

[1] A. Agarwal, P. L. Bartlett, P. Ravikumar, and M. J. Wainwright. Information-Theoretic Lower Bounds on the Oracle Complexity of Stochastic Convex Optimization. *IEEE Transactions on Information Theory*, 58(5):3235–3249, 2012.

[2] Naman Agarwal, Brian Bullins, and Elad Hazan. Second order stochastic optimization in linear time. *arXiv preprint arXiv:1602.03943*, 2016.

[3] Zeyuan Allen Zhu and Elad Hazan. Optimal black-box reductions between optimization objectives. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 1606–1614. 2016.

[4] S.-I. Amari. A theory of adaptive pattern classifiers. *IEEE Transactions on Electronic Computers*, EC-16:299–307, 1967.

[5] S.-I. Amari. Natural gradient works efficiently in learning. *Neural Computation*, 10(2):251–276, 1998.

[6] S.-I. Amari and H. Nagaoka. *Methods of Information Geometry.* American Mathematical Society, Providence, RI, USA, 1997.

[7] Galen Andrew and Jianfeng Gao. Scalable training of l 1-regularized log-linear models. In *Proceedings of the 24th international conference on Machine learning*, pages 33–40. ACM, 2007.

[8] Yves F Atchade, Gersende Fort, and Eric Moulines. On stochastic proximal gradient algorithms. *arXiv preprint arXiv:1402.2365*, 2014.

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

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

[11] S. Becker, E. J. Candès, and M. Grant. Templates for convex cone problems with applications to sparse signal recovery. *Mathematical Programming Computation*, 3(3):165–218, 2011.

[12] S. Becker and Y. LeCun. Improving the convergence of back-propagation learning with second-order methods. In D. Touretzky, G. Hinton, and T. Sejnowski, editors, *Proc. of the 1988 Connectionist Models Summer School*, pages 29–37, San Mateo, 1989. Morgan Kaufman.

[13] D. P. Bertsekas. *Nonlinear Programming.* Athena Scientific, Belmont, Massachusetts, 1995.

[14] D. P. Bertsekas. Incremental least squares methods and the extended Kalman filter. *SIAM Journal on Optimization*, 6(3):807–822, 1996.

[15] D. P. Bertsekas. *Convex Optimization Algorithms.* Athena Scientific, Nashua, NH, USA, 2015.

[16] D. P. Bertsekas and J. N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods.* Prentice-Hall, Englewood Cliffs, NJ, USA, 1989.

[17] D. Blatt, A. O. Hero, and H. Gauchman. A convergent incremental gradient method with a constant step size. *SIAM Journal on Optimization*, 18(1):29–51, 2007.

[18] A. Bordes, L. Bottou, and P. Gallinari. SGD-QN: Careful Quasi-Newton Stochastic Gradient Descent. *Journal of Machine Learning Research*, 10:1737–1754, 2009.

[19] Antoine Bordes, Léon Bottou, Patrick Gallinari, Jonathan Chang, and S. Alex Smith. Erratum: Sgdqn is less careful than expected. *Journal of Machine Learning Research*, 11:2229–2240, 2010.

[20] L. Bottou. Stochastic Gradient Learning in Neural Networks. In *Proceedings of Neuro-Nîmes 91*, Nimes, France, 1991. EC2.

[21] L. Bottou. Online Algorithms and Stochastic Approximations. In D. Saad, editor, *Online Learning and Neural Networks*. Cambridge University Press, Cambridge, UK, 1998.

[22] L. Bottou. Large-Scale Machine Learning with Stochastic Gradient Descent. In Y. Lechevallier and G. Saporta, editors, *Proceedings of the 19th International Conference on Computational Statistics (COMPSTAT'2010)*, pages 177–187, Paris, France, August 2010. Springer.

[23] L. Bottou and O. Bousquet. The Tradeoffs of Large Scale Learning. In J. C. Platt, D. Koller, Y. Singer, and S. T. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 161–168. Curran Associates, Inc., 2008.

[24] L. Bottou, F. Fogelman Soulié, P. Blanchet, and J. S. Lienard. Experiments with Time Delay Networks and Dynamic Time Warping for Speaker Independent Isolated Digit Recognition. In *Proceedings of EuroSpeech 89*, volume 2, pages 537–540, Paris, France, 1989.

[25] L. Bottou and Y. Le Cun. On-line Learning for Very Large Datasets. *Applied Stochastic Models in Business and Industry*, 21(2):137–151, 2005.

[26] O. Bousquet. *Concentration Inequalities and Empirical Processes Theory Applied to the Analysis of Learning Algorithms*. PhD thesis, Ecole Polytechnique, 2002.

[27] C. G. Broyden. The Convergence of a Class of Double-Rank Minimization Algorithms. *Journal of the Institute of Mathematics and Its Applications*, 6(1):76–90, 1970.

[28] R. H. Byrd, G. M. Chin, J. Nocedal, and Y. Wu. Sample Size Selection in Optimization Methods for Machine Learning. *Mathematical Programming, Series B*, 134(1):127–155, 2012.

[29] Richard H Byrd, Gillian M Chin, Will Neveitt, and Jorge Nocedal. On the use of stochastic Hessian information in optimization methods for machine learning. *SIAM Journal on Optimization*, 21(3):977–995, 2011.

[30] Richard H Byrd, Gillian M Chin, Jorge Nocedal, and Figen Oztoprak. A family of second-order methods for convex\ ell _1-regularized optimization. *Mathematical Programming*, pages 1–33, 2012.

[31] Richard H Byrd, Jorge Nocedal, and Figen Oztoprak. An inexact successive quadratic approximation method for convex L-1 regularized optimization. *arXiv preprint arXiv:1309.3529*, 2013.

[32] Emmanuel J Candès and Benjamin Recht. Exact matrix completion via convex optimization. *Foundations of Computational mathematics*, 9(6):717–772, 2009.

[33] Jean-François Cardoso. Blind signal separation: statistical principles. *Proc. of the IEEE*, 9(10):2009–2025, Oct 1998.

[34] N. Cesa-Bianchi, A. Conconi, and C. Gentile. On the Generalization Ability of On-Line Learning Algorithms. *IEEE Transactions on Information Theory*, 50(9):2050–2057, 2004.

[35] Anna Choromanska, Mikael Henaff, Michael Mathieu, Gérard Ben Arous, and Yann LeCun. The loss surfaces of multilayer networks. In *AISTATS*, 2015.

[36] K. L. Chung. On a stochastic approximation method. *Annals of Mathematical Statistics*, 25(3):463–483, 1954.

[37] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. *Trust Region Methods*. Society for Industrial and Applied Mathematics, 2000.

[38] C. Cortes and V. N. Vapnik. Support-Vector Networks. *Machine Learning*, 20(3):273–297, 1995.

[39] R. Courant and H. Robbins. *What is Mathematics?* Oxford University Press, First edition, 1941.

[40] R. Courant and H. Robbins. *What is Mathematics?* Oxford University Press, Second edition, 1996. Revised by I. Stewart.

[41] T. M. Cover. Universal Portfolios. *Mathematical Finance*, 1(1):1–29, 1991.

[42] I. Daubechies, M. Defrise, and C. De Mol. An iterative thresholding algorithm for linear inverse problems with a sparsity constraint. *Communications on Pure and Applied Mathematics*, 58:1413–1457, 2004.

[43] Yann N. Dauphin, Harm de Vries, Junyoung Chung, and Yoshua Bengio. Rmsprop and equilibrated adaptive learning rates for non-convex optimization. *CoRR*, abs/1502.04390, 2015.

[44] Yann N. Dauphin, Razvan Pascanu, Çaglar Gülçehre, KyungHyun Cho, Surya Ganguli, and Yoshua Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 2933–2941, 2014.

[45] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Z. Mao, M. Ranzato, A. W. Senior, P. A. Tucker, K. Yang, and A. Y. Ng. Large Scale Distributed Deep Networks. In *Advances in Neural Information Processing Systems 25*, pages 1232–1240, 2012.

[46] A. Defazio, F. Bach, and S. Lacoste-Julien. SAGA: A Fast Incremental Gradient Method With Support for Non-Strongly Convex Composite Objectives. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 1646–1654. Curran Associates, Inc., 2014.

[47] R. S. Dembo, Eisenstat S. C., and T. Steihaug. Inexact Newton Methods. *SIAM Journal on Numerical Analysis*, 19(2):400–408, 1982.

[48] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977.

[49] Jia Deng, Nan Ding, Yangqing Jia, Andrea Frome, Kevin Murphy, Samy Bengio, Yuan Li, Hartmut Neven, and Hartwig Adam. Large-scale object classification using label relation graphs. In *Proceedings of the 13th European Conference on Computer Vision (ECCV), Part I*, pages 48–64, 2014.

[50] L. Deng, G. E. Hinton, and B. Kingsbury. New types of deep neural network learning for speech recognition and related applications: An overview. In *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2013, Vancouver, BC, Canada*, pages 8599–8603, 2013.

[51] J. E. Dennis and J. J. Moré. A Characterization of Superlinear Convergence and Its Application to Quasi-Newton Methods. *Mathematics of Computation*, 28(126):549–560, 1974.

[52] J. E. Dennis, Jr. and R. B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Classics in Applied Mathematics. SIAM, Philadelphia, PA, USA, 1996.

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

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

[55] R. M. Dudley. *Uniform Central Limit Theorems*. Cambridge Studies in Advanced Mathematics, 63. Cambridge University Press, 1999.

[56] S. T. Dumais, J. C. Platt, D. Hecherman, and M. Sahami. Inductive Learning Algorithms and Representations for Text Categorization. In *Proceedings of the 1998 ACM CIKM International Conference on Information and Knowledge Management, Bethesda, Maryland, USA*, pages 148–155, 1998.

[57] J. Eckstein and D. P. Bertsekas. On the Douglas-Rachford splitting method and the proximal point algorithm for maximal monotone operators. *Mathematical Programming*, 55:293–318, 1992.

[58] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. Liblinear: A library for large linear classification. *The Journal of Machine Learning Research*, 9:1871–1874, 2008.

[59] M. A. T. Figueiredo, R. D. Nowak, and S. J. Wright. Gradient Projection for Sparse Reconstruction: Application to Compressed Sensing and Other Inverse Problems. *IEEE Journal of Selected Topics in Signal Processing*, 1(4):586–597, 2007.

[60] R. Fletcher. A New Approach to Variable Metric Algorithms. *Computer Journal*, 13(3):317–322, 1970.

[61] J. E. Freund. *Mathematical Statistics*. Prentice-Hall, Englewood Cliffs, NJ, USA, 1962.

[62] M. P. Friedlander and M. Schmidt. Hybrid Deterministic-Stochastic Methods for Data Fitting. *SIAM Journal on Scientific Computing*, 34(3):A1380–A1405, 2012.

[63] M. C. Fu. Optimization for simulation: Theory vs. practice. *INFORMS Journal on Computing*, 14(3):192–215, 2002.

[64] D. Gabay and B. Mercier. A Dual Algorithm for the Solution of Nonlinear Variational Problems via Finite Element Approximations. *Computers and Mathematics with Applications*, 2:17–40, 1976.

[65] Gilles Gasso, Aristidis Pappaioannou, Marina Spivak, and Léon Bottou. Batch and online learning algorithms for nonconvex Neyman-Pearson classication. *ACM Transaction on Intelligent System and Technologies*, 3(2), 2011.

[66] E. G. Gladyshev. On stochastic approximations. *Theory of Probability and its Applications*, 10:275–278, 1965.

[67] R. Glowinski and A. Marrocco. Sur l'approximation, par elements finis déordre un, et la resolution, par penalisation-dualité, d'une classe de problems de Dirichlet non lineares. *Revue Française d'Automatique, Informatique, et Recherche Opérationalle*, 9:41–76, 1975.

[68] D. Goldfarb. A Family of Variable Metric Updates Derived by Variational Means. *Mathematics of Computation*, 24(109):23–26, 1970.

[69] G. H. Golub and C. F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, MD, USA, fourth edition, 2012.

[70] Ian J. Goodfellow and Oriol Vinyals. Qualitatively characterizing neural network optimization problems. *CoRR*, abs/1412.6544, 2014.

[71] A. Griewank. *Automatic Differentiation*. Princeton Companion to Applied Mathematics, Nicolas Higham Ed., Princeton University Press, 2014.

[72] Mert Gurbuzbalaban, Asuman Ozdaglar, and Pablo Parrilo. On the convergence rate of incremental aggregated gradient algorithms. *ArXiv*, abs/1506.02081, 2015.

[73] F. S. Hashemi, S. Ghosh, and R. Pasupathy. On adaptive sampling rules for stochastic recursions. In *Simulation Conference (WSC), 2014 Winter*, pages 3959–3970, 2014.

[74] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[75] W. Hoeffding. Probability Inequalities for Sums of Bounded Random Variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

[76] Matthew D. Hoffman, David M. Blei, Chong Wang, and John William Paisley. Stochastic variational inference. *Journal of Machine Learning Research*, 14(1):1303–1347, 2013.

[77] T. Homem-de Mello and A. Shapiro. A Simulation-Based Approach to Two-Stage Stochastic Programming with Recourse. *Mathematical Programming*, 81(3):301–325, 1998.

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

[79] Cho-Jui Hsieh, Mátyás A Sustik, Inderjit S Dhillon, Pradeep K Ravikumar, and Russell Poldrack. Big & quic: Sparse inverse covariance estimation for a million variables. In *Advances in Neural Information Processing Systems*, pages 3165–3173, 2013.

[80] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pages 448–456, 2015.

[81] T. Joachims. Text Categorization with Suport Vector Machines: Learning with Many Relevant Features. In *Proceedings od the 10th European Conference on Machine Learning (ECML'98), Chemnitz, Germany*, pages 137–142, 1998.

[82] Rie Johnson and Tong Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in Neural Information Processing Systems*, pages 315–323, 2013.

[83] R. E. Kalman. A New Approach to Linear Filtering and Prediction Problems. *Transactions of the ASME – Journal of Basic Engineering (Series D)*, 82:35–45, 1960.

[84] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[85] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems 25*, 2012.

[86] Jean Lafond, Nicolas Vasilache, and Léon Bottou. Manuscript in preparation, 2016.

[87] Y. Le Cun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel. Handwritten Digit Recognition with a Back-Propagation Network. In *Advances in Neural Information Processing Systems 2*, pages 396–404, 1989.

[88] Y. Le Cun, L. Bottou, Y. Bengio, and P. Haffner. Gradient Based Learning Applied to Document Recognition. *Proceedings of IEEE*, 86(11):2278–2324, 1998.

[89] Y. Le Cun, L. Bottou, G. B. Orr, and K.-R. Müller. Efficient Backprop. In *Neural Networks, Tricks of the Trade*, Lecture Notes in Computer Science LNCS 1524. Springer Verlag, 1998.

[90] N. Le Roux, M. Schmidt, and F. R. Bach. A Stochastic Gradient Method with an Exponential Convergence Rate for Finite Training Sets. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 2663–2671. Curran Associates, Inc., 2012.

[91] W. S. Lee, P. L. Bartlett, and R. C. Williamson. The Importance of Convexity in Learning with Squared Loss. *IEEE Transactions on Information Theory*, 44(5):1974–1980, 1998.

[92] Todd K. Leen and Genevieve B. Orr. Optimal stochastic search and adaptive momentum. In *Advances in Neural Information Processing Systems 6, [7th NIPS Conference, Denver, Colorado, USA, 1993]*, pages 477–484, 1993.

[93] Dennis Leventhal and Adrian S Lewis. Randomized methods for linear constraints: convergence rates and conditioning. *Mathematics of Operations Research*, 35(3):641–654, 2010.

[94] D. D. Lewis, Y. Yang, T. G. Rose, and F. Li. RCV1: A New Benchmark Collection of Text Categorization Research. *Journal of Machine Learning Research*, 5:361–397, 2004.

[95] Chih-Jen Lin and Jorge J Moré. Newton's method for large bound-constrained optimization problems. *SIAM Journal on Optimization*, 9(4):1100–1127, 1999.

[96] H. Lin, J. Mairal, and Z. Harchaoui. A universal catalyst for first-order optimization. *Advances in Neural Information Processing Systems (NIPS)*, 2015.

[97] D. C. Liu and J. Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, 45:503–528, 1989.

[98] J. Liu, S. J. Wright, C. Ré, V. Bittorf, and S. Sridhar. An Asynchronous Parallel Stochastic Coordinate Descent Algorithm. *Journal of Machine Learning Research*, 16:285–322, 2015.

[99] Gaétan Marceau-Caron and Yann Ollivier. Practical riemannian neural networks. *CoRR*, abs/1602.08007, 2016.

[100] J. Martens and Sutskever I. Learning Recurrent Neural Networks with Hessian-Free Optimization. In *28th International Conference on Machine Learning*. ICML, 2011.

[101] James Martens. New insights and perspectives on the natural gradient method. *arXiv preprint arXiv:1412.1193*, 2014.

[102] P. Massart. Some applications of concentration inequalities to Statistics. *Annales de la Faculté des Sciences de Toulouse*, series 6, 9(2):245–303, 2000.

[103] A. Mokhtari and A. Ribeiro. RES: Regularized Stochastic BFGS algorithm. *IEEE Transactions on Signal Processing*, 62(23):6089–6104, 2014.

[104] N. Murata. A Statistical Study of On-line Learning. In D. Saad, editor, *On-line Learning in Neural Networks*, pages 63–92. Cambridge University Press, New York, NY, USA, 1998.

[105] A. Nemirovski, A. Juditsky, G. Lan, and A. Shapiro. Robust Stochastic Approximation Approach to Stochastic Programming. *SIAM Journal on Optimization*, 19(4):1574–1609, 2009.

[106] A. S. Nemirovski and D. B. Yudin. On Cezaro's convergence of the steepest descent method for approximating saddle point of convex-concave functions. *Soviet Mathematics–Doklady*, 19, 1978.

[107] Y. Nesterov. A method of solving a convex programming problem with convergence rate $\mathcal{O}(1/k^2)$. *Soviet Mathematics Doklady*, 27(2):372–376, 1983.

[108] Y. Nesterov. *Introductory Lectures on Convex Optimization: A Basic Course*. Kluwer Academic Publishers, 2004.

[109] Y. Nesterov. Primal-dual subgradient methods for convex problems. *Mathematical Programming, Series B*, 120(1):221–259, 2009.

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

[111] NIPS Foundation. Advances in neural information processing systems (29 volumes), 1987–2016. Volumes 0–28, http://papers.nips.cc.

[112] F. Niu, B. Recht, C. Re, and S. J. Wright. Hogwild!: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent. In *Advances in Neural Information Processing Systems 24*, pages 693–701, 2011.

[113] J. Nocedal. Updating Quasi-Newton Matrices With Limited Storage. *Mathematics of Computation*, 35(151):773–782, 1980.

[114] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer New York, Second edition, 2006.

[115] A. B. J. Novikoff. On Convergence Proofs on Perceptrons. In *Proceedings of the Symposium on the Mathematical Theory of Automata*, volume 12, pages 615–622, 1962.

[116] J. Nutini, M. Schmidt, I. H. Laradji, M. Friedlander, and H. Koepke. Coordinate Descent Converges Faster with the Gauss-Southwell Rule than Random Selection. In *32nd International Conference on Machine Learning*. ICML, 2015.

[117] J. Ortega and W. Rheinboldt. *Iterative Solution of Nonlinear Equations in Several Variables*. Society for Industrial and Applied Mathematics, 2000.

[118] M. R. Osborne. Fisher's method of scoring. *International Statistical Review / Revue Internationale de Statistique*, 60(1):99–117, Apr 1992.

[119] Hyeyoung Park, Sun-ichi Amari, and Kenji Fukumizu. Adaptive natural gradient learning algorithms for various stochastic models. *Neural Networks*, 13(7):755–764, 2000.

[120] R. Pasupathy, P. W. Glynn, S. Ghosh, and F. S. Hashemi. On Sampling Rates in Stochastic Recursions. 2015. Under review.

[121] B. A. Pearlmutter. Fast Exact Multiplication by the Hessian. *Neural Computation*, 6(1):147–160, 1994.

[122] Mert Pilanci and Martin J Wainwright. Newton sketch: A linear-time optimization algorithm with linear-quadratic convergence. *arXiv preprint arXiv:1505.02250*, 2015.

[123] B. T. Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964.

[124] B. T. Polyak. Comparison of the convergence rates for single-step and multi-step optimization algorithms in the presence of noise. *Engineering Cybernetics*, 15(1):6–10, 1977.

[125] B. T. Polyak. New Method of Stochastic Approximation Type. *Automation and Remote Control*, 51(4):937–946, 1991.

[126] B. T. Polyak and A. B. Juditsky. Acceleration of Stochastic Approximation by Averaging. *SIAM Journal on Control and Optimization*, 30(4):838–855, 1992.

[127] M. J. D. Powell. On Search Directions for Minimization Algorithms. *Mathematical Programming*, 4(1):193–201, 1973.

[128] Maxim Raginsky and Alexander Rakhlin. Information-based complexity, feedback and dynamics in convex programming. *IEEE Transactions on Information Theory*, 57(10):7036–7056, 2011.

[129] A. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson. CNN features off-the-shelf: an astounding baseline for recognition. *arXiv:1403.6382*, 2014.

[130] H. Robbins and S. Monro. A Stochastic Approximation Method. *The Annals of Mathematical Statistics*, 22(3):400–407, 1951.

[131] H. Robbins and D. Siegmund. A convergence theorem for non negative almost supermartingales and some applications. In Jagdish S. Rustagi, editor, *Optimizing Methods in Statistics*. Academic Press, 1971.

[132] Farbod Roosta-Khorasani and Michael W Mahoney. Sub-sampled Newton methods II: Local convergence rates. *arXiv preprint arXiv:1601.04738*, 2016.

[133] Stéphane Ross, Paul Mineiro, and John Langford. Normalized online learning. In *Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence (UAI)*, 2013.

[134] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1, chapter 8, pages 318–362. MIT Press, Cambridge, MA, 1986.

[135] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning Representations by Back-Propagating Errors. *Nature*, 323:533–536, 1986.

[136] D. Ruppert. Efficient estimations from a slowly convergent Robbins-Monro process. Technical report, Cornell University Operations Research and Industrial Engineering, 1988.

[137] Katya Scheinberg and Xiaocheng Tang. Practical inexact proximal quasi-newton method with global complexity analysis. *arXiv preprint arXiv:1311.6547*, 2013.

[138] M. Schmidt. *Graphical Model Structure Learning with ℓ_1-Regularization.* PhD thesis, University of British Columbia, 2010.

[139] M. Schmidt, N. Le Roux, and F. Bach. Minimizing finite sums with the stochastic average gradient. *arXiv preprint arXiv:1309.2388*, 2013.

[140] Mark Schmidt, Nicolas L Roux, and Francis R Bach. Convergence rates of inexact proximal-gradient methods for convex optimization. In *Advances in neural information processing systems*, pages 1458–1466, 2011.

[141] N. N. Schraudolph. Fast Curvature Matrix-Vector Products. In G. Dorffner, H. Bischof, and K. Hornik, editors, *Artificial Neural Networks, ICANN 2001*, volume 2130 of *Lecture Notes in Computer Science*, pages 19–26. Springer-Verlag Berlin Heidelberg, 2001.

[142] N. N. Schraudolph, J. Yu, and S. Günter. A stochastic quasi-Newton method for online convex optimization. In *International Conference on Artificial Intelligence and Statistics*, pages 436–443. Society for Artificial Intelligence and Statistics, 2007.

[143] G. Shafer and V. Vovk. *Probability and finance: It's only a game!*, volume 491. John Wiley & Sons, 2005.

[144] S. Shalev-Shwartz, Y. Singer, N. Srebro, and A. Cotter. Pegasos: primal estimated sub-gradient solver for SVM. *Mathematical Programming*, 127(1):3–30, 2011.

[145] S. Shalev-Shwartz and T. Zhang. Stochastic dual coordinate ascent methods for regularized loss. *Journal of Machine Learning Research*, 14(1):567–599, 2013.

[146] D. F. Shanno. Conditioning of Quasi-Newton Methods for Function Minimization. *Mathematics of Computation*, 24(111):647–656, 1970.

[147] S. Sra, S. Nowozin, and S.J. Wright. *Optimization for Machine Learning*. Mit Pr, 2011.

[148] Stanford Vision Lab. ImageNet Large Scale Visual Recognition Challenge (ILSVRC). `http://image-net.org/challenges/LSVRC/`, 2015 (accessed October 25, 2015).

[149] T. Steihaug. The Conjugate Gradient Method and Trust Regions in Large Scale Optimization. *SIAM Journal on Numerical Analysis*, 20(3):626–637, 1983.

[150] I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. In *30th International Conference on Machine Learning*. ICML, 2013.

[151] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996.

[152] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5. RMSPROP: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012.

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

[154] A. B. Tsybakov. Optimal aggregation of classifiers in statistical learning. *Annals of Statistics*, 32(1), 2004.

[155] V. N. Vapnik. *Estimation of Dependences Based on Empirical Data*. Springer, 1983.

[156] V. N. Vapnik. *Statistical Learning Theory*. Wiley, New York, 1998.

[157] V. N. Vapnik and A. Y. Chervonenkis. *Theory of Pattern Recognition*. Nauka, Moscow, 1974. German Translation: *Theorie der Zeichenerkennung*, Akademie–Verlag, Berlin, 1979.

[158] V. N. Vapnik and A. Ya. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Proceedings of the USSR Academy of Sciences*, 181(4):781–783, 1968. English translation: Soviet Math. Dokl., 9:915-918, 1968.

[159] S. J. Wright, R. D. Nowak, and M. A. T. Figueiredo. Sparse Reconstruction by Separable Approximation. *IEEE Transactions on Signal Processing*, 57(7):2479–2493, 2009.

[160] C. F. J. Wu. On the convergence properties of the em algorithm. *The Annals of Statistics*, 11(1):95–103, 1983.

[161] W. Xu. Towards optimal one pass large scale learning with averaged stochastic gradient descent. *arXiv preprint arXiv:1107.2490v2*, 2011.

[162] Matthew D. Zeiler. ADADELTA: an adaptive learning rate method. *CoRR*, abs/1212.5701, 2012.

[163] C. Zhu, R. H. Byrd, P. Lu, and J. Nocedal. Algorithm 78: L-BFGS-B: Fortran subroutines for large-scale bound constrained optimization. *ACM Transactions on Mathematical Software*, 23(4):550–560, 1997.

[164] M. Zinkevich. Online Convex Programming and Generalized Infinitesimal Gradient Ascent. In *Proceedings of the Twentieth International Conference on Machine Learning (ICML)*, pages 928–936, 2003.