New Optimization Methods for Modern Machine Learning

Sashank J. Reddi

CMU-ML-17-102
July 2017

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

This research was sponsored by: National Science Foundation awards IIS1250350 and IIS1247658; DOE award DESC0011114; and Boeing Aerospace Data Analytics Lab. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.

Keywords: Machine Learning, Optimization, Large-scale, Distributed optimization, Communication-efficient, Finite-sum, Variance-reduction, Bayesian inference

To my parents and my brother.

Abstract

Modern machine learning systems pose several new statistical, scalability, privacy and ethical challenges. With the advent of massive datasets and increasingly complex tasks, scalability has especially become a critical issue in these systems. In this thesis, we focus on fundamental challenges related to scalability, such as computational and communication efficiency, in modern machine learning applications. The underlying central message of this thesis is that classical statistical thinking leads to highly effective optimization methods for modern big data applications.

The first part of the thesis investigates optimization methods for solving large-scale *nonconvex* Empirical Risk Minimization (ERM) problems. Such problems have surged into prominence, notably through deep learning, and have led to exciting progress. However, our understanding of optimization methods suitable for these problems is still very limited. We develop and analyze a new line of optimization methods for nonconvex ERM problems, based on the principle of variance reduction. We show that our methods exhibit fast convergence to stationary points and improve the state-of-the-art in several nonconvex ERM settings, including nonsmooth and constrained ERM. Using similar principles, we also develop novel optimization methods that provably converge to second-order stationary points. Finally, we show that the key principles behind our methods can be generalized to overcome challenges in other important problems such as Bayesian inference.

The second part of the thesis studies two critical aspects of modern distributed machine learning systems — asynchronicity and communication efficiency of optimization methods. We study various asynchronous stochastic algorithms with fast convergence for convex ERM problems and show that these methods achieve near-linear speedups in sparse settings common to machine learning. Another key factor governing the overall performance of a distributed system is its communication efficiency. Traditional optimization algorithms used in machine learning are often ill-suited for distributed environments with high communication cost. To address this issue, we discuss two different paradigms to achieve communication efficiency of algorithms in distributed environments and explore new algorithms with better communication complexity.

Acknowledgments

First and foremost, I would like to express my deepest gratitude to my brilliant advisors, Alex Smola and Barnabás Póczos, for their unwavering support throughout my journey at CMU. Alex Smola has been an exceptional mentor. His sheer brilliance, incredibly sharp insights and amazing expertise in almost every aspect of machine learning has been a great source of inspiration for me. Barnabás Póczos is a brilliant researcher and a wonderful mentor who taught me that patience and persistence is the key to successful research. Also, thanks to my committee members for their constructive feedback throughout this thesis.

I have been extremely fortunate to collaborate with a wonderful set of colleagues: Francis Bach, Emma Brunskill, Carlton Downey, Avinava Dubey, Ahmed Hefny, Jakub Konečný, Barnabás Póczos, Ariel Procaccia, Aaditya Ramdas, Peter Richtárik, Nisarg Shah, Alex Smola, Ruslan Salakhutdinov, Sunita Sarawagi, Aarti Singh, Suvrit Sra, Sundar Vishwanathan, Larry Wasserman, Sinead Williamson, Eric Xing and Manzil Zaheer. Special mention goes to Suvrit Sra, who has been an incredible mentor. His exceptional insights and ideas have been instrumental in shaping a significant part of this thesis. I am forever indebted to my undergraduate advisor Sunita Sarawagi for introducing me to research during my time at IIT Bombay. Also, thanks to my officemates, Avinava, Ahmed, Yifei, Aaditya and Maruan, and department colleagues for countless interesting discussions.

I was lucky enough to have a wonderful set of friends at CMU. Many thanks to my closest friend (read as family), Nisarg. It was a great pleasure discussing research and random subjects, ranging from deep philosophy to public policy, for countless hours with someone as brilliant as Nisarg. His thoughts and insights have been very influential in shaping my career and my life. I would also like to thank Kinjal, Veeru, Yash, Supraja, Jess, Ishan, Avinava and Manzil for being such awesome friends. My time in Pittsburgh would not have been as much fun without their company. I thank all my fellow graduate students for their friendship and support, and for creating a vibrant research environment.

I am very grateful to all my teachers and research collaborators at CMU. I am greatly indebted to Larry Wasserman, Geoff Gordon, Ryan Tibshirani, Aarti Singh and Ruslan Salakhutdinov for sharing their valuable insights and research expertise with me. Also, thanks to the department staff members, especially Diane Stidle and Sandy Winkler, for their invaluable support throughout my time at CMU.

I have been blessed with an amazing family. Special thanks to my sister-in-law, Snigdha, for her constant support and encouragement throughout my doctorate studies. I am lucky to have two wonderful nephews, Arjun and Akhil, who have been a constant source of joy and pride. I would like to thank my grandmothers for their love, wisdom and untold sacrifices. I cannot acknowledge all by name, but let that not diminish my gratitude for the love and support of my uncles, aunts , cousins and rest of the family.

Last but most importantly, I would like to thank my amazing parents, Ravindra Reddy and Girija Kumari, and my brilliant brother, Siddhartha, for their unconditional love, support and countless sacrifices at every moment of my life. Without their guidance and encouragement none of this would have been possible. It is to them I dedicate this thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Machine learning (ML) and intelligent systems have become an indispensable part of our modern society. These systems are now used for a variety of tasks that includes search engines, recommendation engines, self-driving cars and autonomous robots. Most of these systems rely on recognizing patterns in observable data in order to understand the data or make new predictions on unseen data. The advent of modern data collection methods and increased computing machinery have fueled the development of such ML systems. However, modern ML applications also pose new challenges in terms of scalability, efficiency, privacy and ethics; thus, addressing them is critical to the development of the field. This thesis is a step in the direction of addressing these new challenges in modern ML applications.

We start our discussion by further explaining the goals of this thesis. Modern machine learning applications are heavily rooted in statistics and typically involve two major tasks: (i) constructing a model that generates the observable data. (ii) learning the parameters of the model using the observable data. This thesis particularly focuses on *developing fast and efficient mathematical optimization methods to address problem (ii) in modern ML applications*. For the purpose of our discussion, consider the classical problem of classification using a logistic regression classifier. The samples $\{(z_i, y_i)\}_{i=1}^n$ where $z_i \in \mathbb{R}^d$ and $y_i \in \{-1, 1\}$ for all $i \in [n]$ form the dataset where z_i is referred to as features and y_i the corresponding class label. For instance, in email spam filtering task, the training data includes the features z_i corresponding to the content of the email and spam/non-spam label y_i. For such tasks, the optimization problem of our interest is:

$$\min_{x \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^n \log(1 + \exp(-y_i z_i^\top x)) + \frac{1}{2}\|x\|^2. \tag{1.1}$$

The term $\log(1 + \exp(-y_i z_i^\top x))$ represents the loss with respect to the i^{th} sample. The term $\|x\|^2$, referred to as regularizer, improves the quality of the solution by providing better generalization over unseen data. Roughly, this corresponds to finding the maximum likelihood estimate (MLE) of the distribution that explains the observed data. One of the most interesting aspects of this problem is that it is separable over the sample data points. Statistically speaking, such an attribute results from the assumption that the sample points are drawn *i.i.d* from a probability distribution. In modern ML applications, the number of data points n is very large, in which case exploiting the separable nature of the optimization problem becomes important. More generally, in most part of this thesis, we are interested in solving optimization problems of the following form:

$$\min_{x \in \Omega} \frac{1}{n} \sum_{i=1}^{n} f_i(x) + h(x), \tag{1.2}$$

where Ω is a compact convex set. Optimization problems of this form, typically referred to as empirical risk minimization (ERM) problems or finite-sum problems, are central to most applications in ML. For example, in logistic regression problem, $f_i(x) = \log(1 + \exp(-y_i z_i^\top x))$, $h(x) = \frac{1}{2}\|x\|^2$ and $\Omega = \mathbb{R}^d$. In general, for supervised learning tasks, x, f_i and h represent the parameter of our interest, loss with respect to the i^{th} data point and regularization respectively. In most instances, a closed-form solution for problems of the form (1.2) does not exist. Hence, one has to resort to numerical optimization techniques in order to obtain a solution. Numerical optimization methods based on first-order methods (i.e., based on gradient information of the function) are particularly favored in the ML community due to their scalable nature. Popular methods include gradient descent, stochastic gradient descent and randomized coordinate descent. However, as we will see later, these methods can be significantly improved by further exploiting the structure of the problem in (1.2).

Before proceeding any further, one needs to understand the characteristics and requirements of modern machine learning applications in order to appreciate the contributions of this work. Modern ML applications have added the following two new dimensions to the traditional ones.

1. **Increased complexity of the model.** Traditionally, most of the focus in machine learning has been on developing convex models and algorithms (e.g., SVM, logistic regression). However, recently, nonconvex models have surged into the limelight (notably via deep learning) and led to exciting progress – for instance these models have provided state-of-the-art performance and have completely revolutionized areas like computer vision, natural language processing. Thus, developing fast and principled optimization techniques for solving these complex models has become important.

2. **Large-scale and distributed data.** With advances in modern data collection methods, the size of the datasets used in ML applications have increased tremendously. Thus, the dataset is huge and distributed across several computing nodes. For example, large scale distributed machine learning systems such as the *Parameter server* [84], *GraphLab* [176] and *TensorFlow* [1] work with datasets sizes in the order of hundreds of terabytes. When dealing with datasets of such scale in distributed systems, computational and communication workloads need to be designed carefully.

The main focus of this thesis is to make the progress geared towards addressing these important aspects of the modern machine learning applications. Our primary goal is to show that by using principled statistical thinking, one can design very efficient and effective optimization methods with focus on the aforementioned aspects.

1.1 Background

In this section, we briefly review the background material for this thesis. We also describe some common assumptions and notation used for most part of the thesis except where it is mentioned otherwise. The primary optimization problem of our interest is:

$$\min_{x \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} f_i(x), \tag{1.3}$$

where functions f_i can be general *nonconvex* functions. We use empirical risk minimization (ERM) and finite-sum problem interchangeably to describe this problem setting. We use $\mathcal{F}_n$ to denote all functions of the form (1.3). We use component function and training point i interchangeably to refer function f_i for all $i \in [n]$.

We use smoothness assumptions on objective functions throughout this thesis. In particular, we use the following Lipschitz conditions on the function.

Definition 1.1.1. *(L-smooth functions) We say a function $f : \mathbb{R}^d \to \mathbb{R}$ is L-*smooth *if there is a constant L such that $\|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\|, \forall\, x, y \in \mathbb{R}^d$.*
Throughout this thesis, we assume that the functions f_i in (1.3) are L-smooth for all $i \in [n]$. Such an assumption is very common in the analysis of first-order methods. In parts of the thesis, we also examine strongly convex and non-strongly convex functions. A function f is called λ*-strongly convex* if there is $\lambda \geq 0$ such that

$$f(x) \geq f(y) + \langle \nabla f(y), x - y \rangle + \tfrac{\lambda}{2}\|x - y\|^2 \quad \forall x, y \in \mathbb{R}^d.$$

When f is L-smooth and λ-strongly convex, the quantity $\kappa := L/\lambda$ is called the *condition number* of f. We say f is non-strongly convex when f is 0-strongly convex. We also recall the class of gradient dominated functions [115, 122], where a function f is called τ*-gradient dominated* if for any $x \in \mathbb{R}^d$

$$f(x) - f(x^*) \leq \tau \|\nabla f(x)\|^2, \tag{1.4}$$

where x^* is a global minimizer of f. Note that such a function f need not be convex; it is also easy to show that a λ-strongly convex function is $1/(2\lambda)$-gradient dominated. We shall later see that stronger convergence guarantees can be provided for this special class of nonconvex functions. We introduce one more definition useful in the analysis of SGD methods.

Definition 1.1.2. *We say $f \in \mathcal{F}_n$ has a σ-bounded gradient if $\|\nabla f_i(x)\| \leq \sigma$ for all $i \in [n]$ and $x \in \mathbb{R}^d$.*

Finally, the *proximal operator* of a function h is defined as

$$\operatorname{prox}_{\eta h}(x) := \operatorname*{arg\,min}_{y \in \mathbb{R}^d} \left(h(y) + \frac{1}{2\eta} \|y - x\|^2 \right), \tag{1.5}$$

for some parameter $\eta > 0$. Proximal Operators are important for nonsmooth optimization i.e., problems where the objective function is non-differentiable. These operators are particularly useful when (i) the objective function can be written as a sum of smooth and nonsmooth function and (ii) the proximal operator of the nonsmooth part is easy to compute. We defer further discussion on proximal operators to Chapter 5.

1.1.1 Black Box Oracles

We define the black box oracle models used in this thesis. Use of a black box oracle is helpful in providing a clear exposition of the problem and its structure. Furthermore, it also allows us to compare the performance of the various algorithms in a simple manner. We use four different oracle models in this thesis:

1. Incremental First-order Oracle (IFO)
2. Incremental Second-order Oracle (ISO)
3. Proximal Oracle (PO)
4. Linear Oracle (LO)

A majority of the algorithms proposed in this thesis use simple first-order information of the function, and hence, simply query the IFO oracle. We describe only the IFO oracle here and defer the discussion of other oracles to the respective chapters where they have been used.

Definition 1.1.3. *For $f \in \mathcal{F}_n$, an IFO takes an index $i \in [n]$ and a point $x \in \mathbb{R}^d$, and returns the pair $(f_i(x), \nabla f_i(x))$.*

IFO based complexity analysis was introduced to study lower bounds for finite-sum problems [2]. Observe that IFO outputs only first-order information of a specific randomly chosen function f_i. As we shall see later, ISO oracle uses second-order (Hessian) information the function. PO and LO will be useful for nonsmooth and constrained optimization settings.

1.1.2 Convergence Criteria

A significant part of this thesis is devoted to designing and analyzing convergence of optimization algorithms for the aforementioned problem. In the convex case, one typically uses suboptimality conditions based on $[f(x) - f(x^*)]$ or $\|x - x^*\|^2$ as the convergence criterion. Unfortunately, such criteria cannot be used for nonconvex functions due to the hardness of the problem. Following Nesterov [111] and Ghadimi and Lan [47], we use $\|\nabla f(x)\|^2$ to judge when iterate x is approximately stationary. While the quantities $\|\nabla f(x)\|^2$ and $f(x) - f(x^*)$ or $\|x - x^*\|^2$ are not comparable in general (see

[47]), they are typically assumed to be of similar magnitude. Throughout our analysis, we do *not* assume n to be constant, and report dependence on it in our results. For our analysis, we need the following definition.

Definition 1.1.4. *(ϵ-accurate point) A point x is called ϵ-accurate if $\|\nabla f(x)\|^2 \leq \epsilon$. A stochastic iterative algorithm is said to achieve ϵ-accuracy in t iterations if $\mathbb{E}[\|\nabla f(x^t)\|^2] \leq \epsilon$, where the expectation is over the stochasticity of the algorithm.*

In machine learning applications, we are typically interested in generalization error. However, the exact relationship between stationarity and generalization error is unknown in the nonconvex setting and is, thus, out of scope of this thesis. For convex functions, a stronger criterion based on suboptimality can be used. In particular, we use the following convergence criterion for convex and strongly convex functions.

Definition 1.1.5. *(ϵ-suboptimal Point) A point x is called ϵ-suboptimal if $f(x) - f(x^*) \leq \epsilon$, where x^* is an optimal solution of* (1.2). *A stochastic iterative algorithm is said to achieve ϵ-suboptimality in t iterations if $\mathbb{E}[f(x)] - f(x^*) \leq \epsilon$, where the expectation is over the stochasticity of the algorithm.*

We measure efficiency of the algorithms in terms of the number of IFO calls made by the algorithm (IFO complexity) to find an ϵ-accurate or ϵ-suboptimal solution for nonconvex and convex cases respectively, except in Chapter 4, where a stronger convergence criterion for nonconvex functions is considered. Throughout this thesis, we hide the dependence of IFO complexity on the Lipschitz constant L, and the initial point (in terms of $\|x^0 - x^*\|^2$ and $f(x^0) - f(x^*)$) for a clean comparison.

1.2 Overview of Thesis & Our Results

This thesis presents a selection of my work on large-scale optimization methods for machine learning. The content of the thesis is divided into two parts: Part I, consisting of Chapters 2 to 7, describes my work on nonconvex ERM problems, and Part II, organized as the next four chapters, describes my work on large-scale optimization methods with focus on computational and communication efficiency [1]. We provide a brief overview of various chapters here and defer the exact details to the respective chapters. The key message of this thesis is: *Classical statistical thinking leads to highly effective optimization algorithms for modern machine learning applications*.

Part I: Nonconvex Empirical Risk Minimization

The first part of the thesis focuses on fast stochastic methods for nonconvex empirical risk minimization problems. We consider a variety of settings, including nonsmooth and constrained optimization, and provide practical methods with fast convergence rates. We develop a new line of algorithms, called variance reduced algorithms, for nonconvex optimization and show that they can significantly improve the convergence

[1]The code for parts of this thesis is available at https://github.com/CMU-ML-17-102/

of the state-of-the-art algorithms, both from theoretical and practical standpoint.

Chapter 2: Stochastic Variance Reduction for Nonconvex Optimization
Traditionally, for small-scale *nonconvex* optimization problems of form (1.2) that arise in ML, *batch* gradient methods have been used. However, in the large-scale setting i.e., where n is very large in (1.2), batch methods become intractable. In such scenarios, the popular stochastic gradient method (SGD) proposed by Robbins and Monro [148] is often preferred. SGD is an iterative first-order method wherein each step it takes a step in the negative direction of the *stochastic* approximation of the gradient. However, one of the fundamental issues with SGD is the high variance in the stochastic approximation of the gradient slows its convergence. In order to control the variance introduced due to noise in the gradient, one has to typically decrease the step size as the algorithm proceeds. This in turn leads to slow convergence and furthermore, raises the question of selection of step size and its decreasing rate. This limitation has been circumvented in the *convex* settings, where variance reduced variants of SGD with good theoretical and practical properties have been developed. However, the nonconvex setting has been largely unexplored and is still unresolved.

In this chapter, we show that a variant of popular variance reduced method, SVRG, can be used to provide strong convergence guarantees to a stationary point. More specifically, we show that SVRG algorithm can converge to a ϵ-accurate stationary point at a rate $O(n + n^{2/3}/\epsilon)$, faster than both gradient descent (GD) and SGD. Our second contribution of this chapter is analysis of SVRG for *non-strongly convex* functions. Typically, this class of functions is analyzed *indirectly* by first reducing them to the strongly convex functions using l_2 perturbations and then appealing to convergence results for strongly convex functions. However, such a strategy is undesirable in practice due to additional complexity of l_2-perturbation. Instead, we provide direct convergence analysis of SVRG algorithm for this class of functions. Our final contribution of this chapter is linear convergence rate of $O(n + n^{2/3}\tau \log(1/\epsilon))$ to the global optimum for τ-gradient dominated functions. Note that for this class of problems, SGD converges at sublinear convergence rate $O(1/\epsilon^2)$. To our knowledge, ours is the first result that improves upon the convergence rate of SGD and GD for general nonconvex and gradient dominated functions.

Chapter 3: Fast Incremental Methods for Smooth Nonconvex Optimization
Chapter 2 is devoted to investigating convergence of SVRG algorithm for nonconvex ERM problems. Although SVRG exhibits strong theoretical and practical performance for nonconvex optimization, it requires computing *full gradient* at periodic intervals and is thus, not an incremental method. Incremental methods are optimization algorithms that use only constant (independent of n) IFO calls at every iteration of the algorithm. Conceptually, due to very restricted access to component functions, incremental methods are desirable for large-scale empirical risk minimization and more broadly, stochastic optimization. These methods are also appealing from practical point of view since the computational load is distributed uniformly with respect to time.

To this end, we develop fast incremental methods for smooth nonconvex optimization. In particular, we build upon SAGA algorithm, a variance reduced method used in the context of convex optimization, and develop an incremental method with fast convergence rate for nonconvex ERM problems. Perhaps surprisingly, we show that SAGA algorithm converges to ϵ-accurate solution at a rate $O(n + n^{2/3}/\epsilon)$, similar to SVRG algorithm. However, this advantage comes at the cost of additional memory requirement. While such a memory overhead is unavoidable in the general nonconvex setting, we show that for a special class of functions that typically arise in ML, this memory requirement can be reduced. Finally, we also prove that SAGA algorithm exhibits linear convergence for gradient dominated functions. In this chapter, while we confine our attention to SAGA algorithm, by using our proof technique, we can show that another incremental method, SDCA, exhibits very similar properties for our general problem setting.

Chapter 4: A Generic Approach for Escaping Saddle Points
The previous two chapters focus on understanding the convergence of stochastic methods, like SVRG and SAGA, to *stationary* points of nonconvex functions i.e., first-order critical points. While these results significantly advance our understanding, they have an important shortcoming — they do not ensure convergence to local optima or second-order critical points. This issue has recently received considerable attention in the ML community, especially in the context of deep learning [27, 29, 30], who argue that convergence to saddle points is the primary obstacle for solving nonconvex ERM problems like large deep networks.

In this chapter, we tackle this fundamental issue of convergence to second-order critical points. To ensure such a property, one has to resort to algorithms that either use the Hessian explicitly or exploit its structure. Thus, this chapter, unlike other chapters of this thesis, uses second-order information of the objective function. However, frequent usage of second-order information drastically increases the computational and memory requirements of the system; thereby, reducing its overall performance. For example, cubic regularization (CR) method [115] uses Hessians to obtain faster convergence rates. In particular, Nesterov and Polyak [115] showed that CR requires $O(1/\epsilon^{3/2})$ iterations to achieve the second-order critical conditions. However, each iteration of CR is expensive requiring computation of the Hessian and solving multiple linear systems, each of which has complexity $O(d^{\omega})$ (ω is the matrix multiplication constant), thus, undermining the benefit of its faster convergence. The insight of our work is that first-order information can be used for the most part of the optimization process and Hessian information is needed only when stuck at stationary points that are not second-order critical; thus, eliminating the need for frequent use of Hessian information.

Our first contribution in this chapter is to develop a general framework based on this methodology, one which carefully alternates between subroutines that use gradient and Hessian information respectively and ensures second-order criticality, and provide its convergence analysis. We provide two different instantiations of the framework and show that a simple instantiation of the framework achieves convergence rates that are competitive to the current state-of-the-art results.

Chapter 5: Fast Stochastic Methods for Nonsmooth Nonconvex Optimization

Nonsmoothness arises naturally in ML settings as part of the loss function or the regularization. For example, in support vector machines (SVM), nonsmoothness is due to the non-differentiability of the hinge loss. This is just one of several instances in ML where the objective function is nonsmooth. In this chapter, we focus on fast stochastic methods for nonsmooth nonconvex optimization problems, thereby, generalizing the problem settings in the previous chapters. One of the popular approach to handle nonsmoothness is through the use of proximal operators. Proximal operators can be used in problem settings of the following form:

$$\min_x f(x) + h(x), \tag{1.6}$$

where function f is smooth and h is nonsmooth and proximal operator of function h can be computed (see (1.5)). Such a setting is very natural to ML, where function h corresponds to the nonsmooth regularizer. Proximal variants of SGD algorithm has been extensively studied in the context of convex optimization and been recently generalized to the nonconvex setting.

In the convex case, proximal variants of algorithms are appealing because they have convergence rates very similar to that of their smooth counterparts. For instance, proximal variant of SGD algorithm has convergence rate $O(1/\epsilon^2)$ to obtain an ϵ-suboptimal solution, which is similar to that of SGD for smooth convex optimization. However, it is unknown if such a benefit extends to the nonconvex setting. In a recent work, [48] showed that proximal variant of SGD algorithm converges in the setting where f is smooth nonconvex and r is nonsmooth but convex; however, it requires increasing the mini-batch size at each iteration of the algorithm. In particular, it is unknown if the constant mini-batch proximal variant of SGD converges for the nonconvex case. Note that this issue does not afflict the smooth nonconvex setting where SGD is known to converge with constant mini-batch size. Our primary contribution in this chapter is to develop and study fast stochastic methods for tackling nonsmooth nonconvex problems with guaranteed convergence for constant minibatches; thereby, bridging a fundamental gap in our knowledge of stochastic nonsmooth nonconvex optimization. These methods are based on proximal variants of the variance reduced methods, SVRG and SAGA, once again demonstrating the power of reducing variance in the context of nonconvex optimization.

Chapter 6: Projection-Free Stochastic Nonconvex Optimization

In the previous chapter, the primary focus was on handling nonsmoothness through the use of proximal operators, which includes constrained optimization as a special case. Consider the following optimization problem:

$$\min_{x \in \Omega} \frac{1}{n} \sum_{i=1}^{n} f_i(x),$$

where set Ω is convex. This is a special case of (1.6), where function h corresponds to the *indicator function* $\mathcal{I}_\Omega(x)$ of a closed convex set Ω, defined below:

$$\mathcal{I}_\Omega(x) = \begin{cases} 0 & \text{when } x \in \Omega \\ +\infty & \text{when } x \notin \Omega. \end{cases}$$

An important assumption in this context is that the proximal operator of r is easy to compute. In the constrained optimization setting, this amounts to efficient projection onto the constraint set. While such an assumption is reasonable in several ML applications, in many real settings, the cost projecting onto the constraints set can be very high (e.g., projecting onto the trace-norm ball, onto base polytopes in submodular minimization [42]); and in extreme cases projection can even be computationally intractable [28]. This fundamental issue underlies the recent surge in interest for projection-free methods. Frank-Wolfe (FW) method, also known as conditional gradient, is a classic projection-free method developed for constrained convex optimization. More recently, stochastic FW methods have been developed for ML applications since batch FW method is infeasible for large-scale ML applications because it requires access to all component functions at each iteration of the algorithm; however, nonconvex variants of these methods are unknown. Developing nonconvex projection-free methods is critical for modern large-scale ML applications, especially in wake of recent breakthroughs in deep learning.

To address this challenge, following the line of work on variance reduced methods in thesis, we propose two variance reduced (VR) algorithms: SVFW and SAGAFW, based on SVRG and SAGA respectively. By careful selection of parameters in these algorithms, we can attain faster convergence rates than the deterministic FW. In particular, we prove that SVFW and SAGAFW are faster than deterministic FW by a factor of $n^{1/3}$ and $n^{2/3}$. In this chapter, we also investigate the purely stochastic setting and show that significant improvements over out-of-the-box stochastic FW method for this setting. To our knowledge, our work presents the first theoretical improvement for stochastic variants of projection-free methods, like Frank-Wolfe, in the context of nonconvex optimization.

Chapter 7: Variance Reduced Stochastic Langevin Dynamics
Bayesian inference (BI) is a very closely related to optimization. BI involves sampling from the posterior distribution and exploring its landscape, which is intimately related to finding the minimum of a function. Our focus in this chapter is to show that variance reduction techniques can be useful in settings beyond optimization.

Monte Carlo based sampling methods are preferred in Bayesian inference due to their simplicity and asymptotic convergence properties; however convergence can be slow in large models due to poor mixing. This led to the rise of gradient-based Monte Carlo methods like Langevin Dynamics and Hamiltonian Monte Carlo [104], which allow more efficient exploration of posterior distributions. However, the practical performance of these methods is greatly undermined by the huge computational cost of computing the gradient and evaluating the likelihood on large datasets. More recently, stochastic variants of these methods, replace the expensive gradient evaluation with a cheap stochastic gradient approximation on a random subset of the data, have garnered significant interest while working with large data sets, especially Stochastic Gradient

Langevin Dynamics (SGLD) [172]. SGLD algorithm has very close relationship with classic SGD method [149]. Similar to SGD, performance of SGLD can be hampered by the variance in the stochastic gradient approximation, thereby, leading to slow convergence of the algorithm. Hence, it is natural to ask if SGLD, similar to SGD, can benefit from variance reduction techniques.

We provide an affirmative answer to the aforementioned research question by developing new methods for reducing variance in the stochastic approximation in SGLD. In particular, we propose two different methods, SVRG-LD and SAGA-LD inspired from SVRG and SAGA algorithms respectively, that use variance reduced gradient approximation. For these methods, we provide strong theoretical results, and empirical evaluations showing impressive speed-ups over SGLD, on a variety of machine learning tasks such as regression, classification, independent component analysis and mixture modeling. To our knowledge, ours is the first work that aims to directly reduce variance in a gradient-based Monte Carlo method. While our focus in this chapter is on Langevin dynamics, our approach is easily applicable to other gradient-based Monte Carlo methods.

Part II: Large-Scale Asynchronous & Distributed Optimization

In the second part, we look into asynchronous and distributed empirical risk minimization problems. In particular, we assume a setting where parallelism is important and communication between the nodes is expensive. In order to meet these requirements, the distributed optimization algorithm needs to be robust in terms of (i) synchronicity and (ii) the communication load of the overall system. Our focus in this part of the thesis is to examine these vital aspects, and design novel asynchronous and communication efficient algorithms for *convex* ERM problems.

Chapter 8: Asynchronous Stochastic Variance Reduced Algorithms for ERM
Most traditional ML algorithms, like SGD, are inherently sequential. For instance, the update of SGD,

$$x^{t+1} = x^t - \frac{1}{|I_t|} \sum_{i \in I_t} \nabla f_i(x^t)$$

for a uniformly randomly chosen subset $I_t \subset [n]$, is sequential requiring computation of gradient only part of the training data. In parallel computing environments, this results in only partial usage of computing resources, which can immensely damage the performance of the system; hence, asynchronous systems are vital to many modern large-scale ML systems. Recently, several works, most notably Hogwild! [128], have demonstrated that SGD algorithm can be implemented in an asynchronous fashion at the expense of very little practical and theoretical efficiency. However, these variants of SGD inherit the same slow convergence of SGD.

VR methods, like SVRG and SAGA, have been able to overcome these problems in the synchronous setting, however, asynchronous versions of these algorithms—a crucial requirement for modern large-scale applications—have not been studied. We bridge this gap by presenting a new unifying framework for many variance reduction techniques. Subsequently, we propose an asynchronous algorithm grounded in our framework, and prove its fast convergence. An important consequence of our general approach is that it yields asynchronous versions of variance reduction algorithms such as SVRG and SAGA as a byproduct, which exhibit linear convergence to the global minimum for strongly-convex functions. Our method achieves near linear speedup in sparse settings common to machine learning. We demonstrate the empirical performance of our method through a concrete realization of asynchronous SVRG. To our knowledge, ours is the first work to develop *asynchronous* stochastic methods with linear convergence for strongly-convex functions.

Chapter 9: Asynchronous Randomized Coordinate Descent Algorithms for ERM
In this chapter, we develop asynchronous variants for randomized coordinate descent with linear constraints. In this setting, we are interested in the following composite objective convex problem with *non-separable* linear constraints

$$\min_x F(x) := f(x) + h(x) \quad \text{s.t.} \quad Ax = 0. \tag{1.7}$$

Here, $f : \mathbb{R}^d \to \mathbb{R}$ is assumed to be continuously differentiable and convex, while $h : \mathbb{R}^d \to \mathbb{R} \cup \{\infty\}$ is lower semi-continuous, convex, coordinate-wise separable, but not necessarily smooth; the linear constraints are specified by a matrix $A \in \mathbb{R}^{m \times d}$, for which $m \ll d$. One might wonder the connection between optimization problem (1.7) and finite-sum minimization problems of our interest. However, observe that any finite-sum minimization problem can be rewritten using variable-splitting in the following manner:

$$\min_x \frac{1}{n} \sum_{i=1}^{n} f_i(x) \equiv \min_{\{x_i = x, \forall i \in [n]\}} \frac{1}{n} \sum_{i=1}^{n} f_i(x_i).$$

Solving the problem in distributed environment requires considerable synchronization (for the consensus constraint), which can slow down the algorithm significantly. However, the dual of the problem is

$$\min_\lambda \frac{1}{n} \sum_{i=1}^{n} f_i^*(\lambda_i) \quad s.t \ \sum_{i=1}^{n} \lambda_i = 0,$$

where f_i^* is the Fenchel conjugate of f_i. This reformulation perfectly fits our problem formulation in (1.7) and can be solved in an asynchronous manner using the proposed procedure. Other interesting applications of the more general setup include constrained least square problem, multi-agent planning problems, resource allocation—see [105, 106] and references therein for more examples. We also demonstrate the superior empirical performance of our algorithm on synthetic and real-world applications.

Chapter 10: Communication-Efficient Coresets for Empirical Risk Minimization
In the previous chapters of Part II, we discuss asynchronous optimization algorithms for problems that arise in machine learning. However, the communication efficiency

of these algorithms was not examined. In modern distributed ML systems, machines read and write global parameter frequently. This data access requires massive amount of network bandwidth and hence, communication efficiency of optimization algorithms is critical. To this end, we develop and discuss new communication efficient optimization algorithms. For the purpose of this discussion, we assume a parameter server architecture for investigating these algorithms. Such a setup entails a server group and a worker group where each group contains several threads/machines. The server machines mainly serve the purpose of maintaining the global parameters, while most of the workload needs to be allocated to the worker machines. The communication between the worker and the server group is the assumed to the main bottleneck and hence, needs to minimized.

Iterative algorithms like SGD often require large amount of communication between worker and server machines. The first approach to reduce this communication cost is by constructing a small summary of the training data — which acts as a proxy for the entire data set — and communicating it to the server machine; thereby, eliminating the need for frequent communication between the worker and the server machines. Such an approach entails (i) computing these summaries of data at the worker nodes (which can be computationally intensive), (ii) sending these *small* summaries to a server machine (a low communication overhead task) and (iii) finally, solving a *small* optimization problem at the server machines. This summary of the training points is called a *coreset*. While this methodology has been successfully applied to data clustering problems like k-means and k-median (see [38, 39]), it remains largely unexplored for supervised learning and optimization problems. In this chapter, we investigate this methodology in the context of solving empirical loss minimization problems and show its benefit for ERM problems with particular loss functions like hinge and logistic loss.

Chapter 11: Communication-Efficient Distributed Optimization for ERM
An alternate approach to attain communication efficiency is to perform most of the computation at the worker machines in an embarrassingly manner and combine the solutions from all the worker machines; consequently, eliminating the need to communicate to the server machine frequently. Note that such an operation needs to be done in an iterative fashion in order to each an optimal solution to our optimization problem. ADMM (alternating direction method of multipliers) is an popular approach for solving distribution optimization problems that follows this methodology. Under certain conditions, ADMM is shown to achieve communication complexity of $O(\sqrt{L/\lambda}\log(1/\epsilon))$ for L-smooth and λ-strongly convex function. More recently, the DANE, DISCO and COCOA+ algorithms have been proposed to tackle the problem of reducing the communication complexity in solving problems of form (1.2) [67, 163, 180]. DISCO is particularly appealing because they match communication complexity lower bounds derived in [175]. However, DISCO requires a second-order oracle for its execution and is not embarrassingly parallel. Our contribution in this chapter is to develop a *first-order* algorithm that not only achieves the communication lower bounds in [175] but can be also be implemented in an embarrassingly parallel fashion. This algorithm, AIDE, is to our knowledge the first gradient-based method that achieves the lower bounds in [175].

1.3 Bibliographic Notes

The research presented in this thesis is based on joint work with many co-authors, as described below. In each work, I am either the primary contributor or one of two equal primary contributors.

In Part I, Chapter 2 is based on joint work with Ahmed Hefny, Suvrit Sra, Barnabás Póczos and Alex Smola [139]. Chapter 3, 5 and 6 are joint works with Suvrit Sra, Barnabás Póczos and Alex Smola [141, 142, 143]. Chapter 4 is joint work with Manzil Zaheer, Suvrit Sra, Barnabás Póczos, Francis Bach, Ruslan Salakhutdinov and Alex Smola. Chapter 7 is joint work with Avinava Dubey, Sinead Williamson, Barnabás Póczos, Alex Smola and Eric Xing [138].

In Part II, Chapter 8 is based on joint work with Ahmed Hefny, Suvrit Sra, Barnabás Póczos and Alex Smola [133]. Chapter 9 is joint work with Ahmed Hefny, Carlton Downey, Avinava Dubey and Suvrit Sra [132]. Chapter 10 is joint work with Barnabás Póczos and Alex Smola [134]. Chapter 11 is joint work with Jakub Konečný, Peter Richtárik, Barnabás Póczos and Alex Smola [140].

1.3.1 Excluded Research

This thesis not does include a portion of my work during my PhD. Chapter 12 provides an overview of my research excluded from this thesis. Most of these works focus on statistical aspects of machine learning. The excluded research includes:

- My work on decision processes: [129].
- My work on doubly robust estimation: [135].
- My work on functional data analysis: [131].
- My work on kernel methods and hypothesis testing: [126, 130, 136, 137].

1.4 How to read this Thesis?

The introduction chapter provides a brief overview of all the chapters and relationship between results of various chapters, and is a prerequisite to rest of the thesis. Each chapter is self-contained and independent of other chapters so that it can be read without any reference to rest of this thesis. Furthermore, the reader is not assumed to have detailed knowledge of machine learning and optimization; we rather try to provide the necessary knowledge, allowing the thesis to be accessible to graduate students.

Part I

Nonconvex Empirical Risk Minimization

Chapter 2

Stochastic Variance Reduction for Smooth Nonconvex Optimization

2.1 Introduction

In this chapter, we investigate fast stochastic methods for non-convex finite-sum problems. In particular, we study nonconvex *finite-sum* problems of the form

$$\min_{x \in \mathbb{R}^d} f(x) := \frac{1}{n} \sum_{i=1}^{n} f_i(x), \tag{2.1}$$

where neither f nor the individual f_i ($i \in [n]$) are necessarily convex; just Lipschitz smooth (i.e., Lipschitz continuous gradients). Problems of this form arise naturally in ML in the form of empirical risk minimization (ERM). Algorithms that use IFOs are favored for these problems as they require only a small amount first-order information at each iteration. Two fundamental models in machine learning that profit from IFO algorithms are (i) empirical risk minimization, which typically uses convex finite-sum models; and (ii) deep learning, which uses nonconvex ones.

The prototypical IFO algorithm, stochastic gradient descent (SGD)[1] has witnessed tremendous progress in the recent years. By now a variety of accelerated, parallel, and faster converging versions are known. Among these, of particular importance are variance reduced (VR) stochastic methods [33, 71, 153], which have delivered exciting progress such as linear convergence rates (for strongly convex functions) as opposed to sublinear rates of ordinary SGD [110, 148]. Similar (but not same) benefits of VR methods can also be seen in smooth convex functions. The SVRG algorithm of [71] is particularly attractive here because of its low storage requirement in comparison to the algorithms in [33, 153].

Despite the meteoric rise of VR methods, their analysis for general nonconvex problems is largely missing. [71] remark on convergence of SVRG when $f \in \mathcal{F}_n$ is locally

[1]We use 'incremental gradient' and 'stochastic gradient' interchangeably, though we are only interested in finite-sum problems.

Algorithm	Nonconvex	Convex	Gradient Dominated	Fixed Step Size?
SGD	$O(1/\epsilon^2)$	$O(1/\epsilon^2)$	$O(1/\epsilon^2)$	$\times$
GD	$O(n/\epsilon)$	$O(n/\epsilon)$	$O(n\tau\log(1/\epsilon))$	$\surd$
SVRG	$O(n + (n^{2/3}/\epsilon))$	$O(n + (\sqrt{n}/\epsilon))$	$O((n + n^{2/3}\tau)\log(1/\epsilon))$	$\surd$

Table 2.1: Table comparing the *best* IFO complexity of different algorithms discussed in this chapter. The complexity is measured in terms of the number of oracle calls required to achieve an ϵ-accurate solution (see Definition 2.2.1). Here, by fixed step size, we mean that the step size of the algorithm is fixed and does not dependent on ϵ (or alternatively T, the total number of iterations). The complexity of gradient dominated functions refers to the number of IFO calls required to obtain ϵ-accurate solution for a τ-gradient dominated function (see Section 2.2 for the definition)

strongly convex and provide compelling experimental results (Fig. 4 in [71]). However, problems encountered in practice are typically not even locally convex, let alone strongly convex. The current analysis of SVRG does not extend to nonconvex functions as it relies heavily on convexity for controlling the variance. Given the dominance of stochastic gradient methods in optimizing deep neural nets and other large nonconvex models, theoretical investigation of faster nonconvex stochastic methods is much needed.

Convex VR methods are known to enjoy the faster convergence rate of GD but with a much weaker dependence on n, without compromising the rate like SGD. However, it is not clear if these benefits carry beyond convex problems, prompting the central question of this chapter:

For nonconvex functions in $\mathcal{F}_n$, can one achieve convergence rates faster than both SGD *and* GD *using an IFO? If so, then how does the rate depend on n and on the number of iterations performed by the algorithm?*

Perhaps surprisingly, we provide an affirmative answer to this question by showing that a careful selection of parameters in SVRG leads to faster convergence than both GD and SGD. To our knowledge, ours is the *first* work to improve convergence rates of SGD and GD for IFO-based nonconvex optimization. The key complexity results are listed in Table 2.1.

2.1.1 Related Work

Convex. Bertsekas [18] surveys several incremental gradient methods for convex problems. A key reference for stochastic convex optimization (for $\min \mathbb{E}_z[F(x,z)]$) is [110]. Faster rates of convergence are attained for problems in $\mathcal{F}_n$ by VR methods, see e.g., [33, 34, 71, 75, 153, 156]. Asynchronous VR frameworks are developed in [133]. Agarwal and Bottou [2], Lan and Zhou [81] study lower-bounds for convex finite-sum problems. Shalev-Shwartz [154] prove linear convergence of stochastic dual coordinate ascent when the individual f_i ($i \in [n]$) are nonconvex but f is strongly convex. They do not study the general nonconvex case. Moreover, even in their special setting our results improve upon theirs for the high condition number regime.

Nonconvex. SGD dates at least to the seminal work [148]; and since then it has been developed in several directions [19, 77, 93, 124]. In the (nonsmooth) finite-sum setting, Sra [164] considers proximal splitting methods, and analyzes asymptotic convergence with nonvanishing gradient errors. Hong [60] studies a distributed nonconvex incremental ADMM algorithm.

These works, however, only prove expected convergence to stationary points and often lack analysis of rates. The first nonasymptotic convergence rate analysis for SGD is in [47], who show that SGD ensures $\|\nabla f\|^2 \leq \epsilon$ in $O(1/\epsilon^2)$ iterations. A similar rate for parallel and distributed SGD was shown recently in [87]. GD is known to ensure $\|\nabla f\|^2 \leq \epsilon$ in $O(1/\epsilon)$ iterations [111, Chap. 1.2.3].

The first analysis of nonconvex SVRG seems to be due to Shamir [160], who considers the special problem of computing a few leading eigenvectors (e.g., for PCA); see also the follow up work [161]. Finally, we note another interesting example, stochastic optimization of locally quasi-convex functions [57], wherein actually a $O(1/\epsilon^2)$ convergence in function value is shown.

2.2 Background & Problem Setup

We say f is *L-smooth* if there is a constant L such that

$$\|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\|, \quad \forall\, x, y \in \mathbb{R}^d.$$

Throughout, we assume that the functions f_i in (2.1) are L-smooth, so that $\|\nabla f_i(x) - \nabla f_i(y)\| \leq L\|x - y\|$ for all $i \in [n]$. Such an assumption is very common in the analysis of first-order methods. A function f is called *λ-strongly convex* if there is $\lambda \geq 0$ such that

$$f(x) \geq f(y) + \langle \nabla f(y), x - y \rangle + \tfrac{\lambda}{2}\|x - y\|^2 \quad \forall x, y \in \mathbb{R}^d.$$

The quantity $\kappa := L/\lambda$ is called the *condition number* of f, whenever f is L-smooth and λ-strongly convex. We say f is non-strongly convex when f is 0-strongly convex.

We also recall the class of gradient dominated functions [115, 122], where a function f is called *τ-gradient dominated* if for any $x \in \mathbb{R}^d$

$$f(x) - f(x^*) \leq \tau \|\nabla f(x)\|^2, \tag{2.2}$$

where x^* is a global minimizer of f. Note that such a function f need not be convex; it is also easy to show that a λ-strongly convex function is $1/2\lambda$-gradient dominated.

We analyze convergence rates for the above classes of functions. Following Nesterov [111] and Ghadimi and Lan [47], we say an iterate x is approximately stationary if $\|\nabla f(x)\|^2 \leq \epsilon$. Contrast this with SGD for convex f, where one uses $[f(x) - f(x^*)]$ or $\|x - x^*\|^2$ as a convergence criterion. Unfortunately, such criteria cannot be used for nonconvex functions due to the hardness of the problem. While the quantities $\|\nabla f(x)\|^2$

and $f(x) - f(x^*)$ or $\|x - x^*\|^2$ are not comparable in general (see [47]), they are typically assumed to be of similar magnitude. Throughout our analysis, we do *not* assume n to be constant, and report dependence on it in our results. For our analysis, we need the following definition.

Definition 2.2.1. *A point x is called ϵ-accurate if $\|\nabla f(x)\|^2 \le \epsilon$. A stochastic iterative algorithm is said to achieve ϵ-accuracy in t iterations if $\mathbb{E}[\|\nabla f(x^t)\|^2] \le \epsilon$, where the expectation is over the stochasticity of the algorithm.*

We measure the efficiency of the algorithms in terms of the number of IFO calls made by the algorithm (IFO complexity) to achieve an ϵ-accurate solution. Throughout the chapter, we hide the dependence of IFO complexity on Lipschitz constant L, and the initial point (in terms of $\|x^0 - x^*\|^2$ and $f(x^0) - f(x^*)$) for a clean comparison. We introduce one more definition useful in the analysis of SGD methods for bounding the variance.

Definition 2.2.2. *We say $f \in \mathcal{F}_n$ has a σ-bounded gradient if $\|\nabla f_i(x)\| \le \sigma$ for all $i \in [n]$ and $x \in \mathbb{R}^d$.*

2.2.1 Nonconvex SGD: Convergence Rate

Stochastic gradient descent (SGD) is one of the simplest algorithms for solving (2.1); Algorithm 1 lists its pseudocode. By using a uniformly randomly chosen (with re-

Algorithm 1: SGD

```
1: Input: $x^0 \in \mathbb{R}^d$, Step-size sequence: $\{\eta_t > 0\}_{t=0}^{T-1}$
2: for $t = 0$ to $T - 1$ do
3:     Uniformly randomly pick $i_t$ from $\{1, \dots, n\}$
4:     $x^{t+1} = x^t - \eta_t \nabla f_{i_t}(x)$
5: end for
```

placement) index i_t from $[n]$, SGD uses an unbiased estimate of the gradient at each iteration. Under appropriate conditions, Ghadimi and Lan [47] establish convergence rate of SGD to a stationary point of f. Their results include the following theorem.

Theorem 2.2.3. *Suppose f has σ-bounded gradient; let $\eta_t = \eta = c/\sqrt{T}$ where $c = \sqrt{\frac{2(f(x^0)-f(x^*))}{L\sigma^2}}$, and x^* is an optimal solution to* (2.1). *Then, the iterates of Algorithm 1 satisfy*

$$\min_{0 \le t \le T-1} \mathbb{E}[\|\nabla f(x^t)\|^2] \le \sqrt{\frac{2(f(x^0) - f(x^*))L}{T}}\sigma.$$

For completeness we present a proof in the appendix. Note that our choice of step size η requires knowing the total number of iterations T in advance. A more practical approach is to use a $\eta_t \propto 1/\sqrt{t}$ or $1/t$. A bound on IFO calls made by Algorithm 1 follows as a corollary of Theorem 2.2.3.

Corollary 2.2.3.1. *Suppose function f has σ-bounded gradient, then the IFO complexity of Algorithm 1 to obtain an ϵ-accurate solution is $O(1/\epsilon^2)$.*

As seen in Theorem 2.2.3, SGD has a convergence rate of $O(1/\sqrt{T})$. This rate is not improvable in general even when the function is (non-strongly) convex [108]. This barrier is due to the variance introduced by the stochasticity of the gradients, and it is not clear if better rates can be obtained SGD even for convex $f \in \mathcal{F}_n$.

2.3 Nonconvex SVRG

We now turn our focus to variance reduced methods. We use SVRG [71], an algorithm recently shown to be very effective for reducing variance in convex problems. As a result, it has gained considerable interest in both machine learning and optimization communities. We seek to understand its benefits for *nonconvex* optimization. For reference, Algorithm 2 presents SVRG's pseudocode.

Observe that Algorithm 2 operates in epochs. At the end of epoch s, a full gradient is calculated at the point $\tilde{x}^s$, requiring n calls to the IFO. Within its inner loop SVRG performs m stochastic updates. The total number of IFO calls for each epoch is thus $\Theta(m+n)$. For $m = 1$, the algorithm reduces to the classic GD algorithm. Suppose m is chosen to be $O(n)$ (typically used in practice), then the total IFO calls per epoch is $\Theta(n)$. To enable a fair comparison with SGD, we assume that the total number of inner iterations across all epochs in Algorithm 2 is T. Also note a simple but important implementation detail: as written, Algorithm 2 requires storing all the iterates x_t^{s+1} $(0 \le t \le m)$. This storage can be avoided by keeping a running average with respect to the probability distribution $\{p_i\}_{i=0}^m$.

Algorithm 2 attains linear convergence for strongly convex f [71]; for non-strongly convex functions, rates faster than SGD can be shown by using an indirect perturbation argument—see e.g., [74, 173].

We first state an intermediate result for the iterates of nonconvex SVRG. To ease exposition, we define

$$\Gamma_t = \left(\eta_t - \frac{c_{t+1}\eta_t}{\beta_t} - \eta_t^2 L - 2c_{t+1}\eta_t^2\right), \tag{2.3}$$

for some parameters c_{t+1} and β_t (to be defined shortly).

Our first main result is the following theorem that provides convergence rate of Algorithm 2.

Theorem 2.3.1. *Let $f \in \mathcal{F}_n$. Let $c_m = 0$, $\eta_t = \eta > 0$, $\beta_t = \beta > 0$, and $c_t = c_{t+1}(1 + \eta\beta + 2\eta^2 L^2) + \eta^2 L^3$ such that $\Gamma_t > 0$ for $0 \le t \le m-1$. Define the quantity $\gamma_n := \min_t \Gamma_t$. Further, let $p_i = 0$ for $0 \le i < m$ and $p_m = 1$, and let T be a multiple of m. Then for the output x_a of Algorithm 2 we have*

$$\mathbb{E}[\|\nabla f(x_a)\|^2] \le \frac{f(x^0) - f(x^*)}{T\gamma_n},$$

where x^* is an optimal solution to (2.1)*.*

Algorithm 2: SVRG$\left(x^0, T, m, \{p_i\}_{i=0}^{m}, \{\eta_i\}_{i=0}^{m-1}\right)$

```
1: Input: $\tilde{x}^0 = x_m^0 = x^0 \in \mathbb{R}^d$, epoch length $m$, step sizes $\{\eta_i > 0\}_{i=0}^{m-1}$, $S = \lceil T/m \rceil$, discrete
   probability distribution $\{p_i\}_{i=0}^{m}$
2: for $s = 0$ to $S - 1$ do
3:     $x_0^{s+1} = x_m^s$
4:     $g^{s+1} = \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(\tilde{x}^s)$
5:     for $t = 0$ to $m - 1$ do
6:         Uniformly randomly pick $i_t$ from $\{1, \dots, n\}$
7:         $v_t^{s+1} = \nabla f_{i_t}(x_t^{s+1}) - \nabla f_{i_t}(\tilde{x}^s) + g^{s+1}$
8:         $x_{t+1}^{s+1} = x_t^{s+1} - \eta_t v_t^{s+1}$
9:     end for
10:    $\tilde{x}^{s+1} = \sum_{i=0}^{m} p_i x_i^{s+1}$
11: end for
12: Output: Iterate $x_a$ chosen uniformly random from $\{\{x_t^{s+1}\}_{t=0}^{m-1}\}_{s=0}^{S-1}$.
```

Furthermore, we can also show that nonconvex SVRG exhibits expected descent (in objective) after every epoch. The condition that T is a multiple of m is solely for convenience and can be removed by slight modification of the theorem statement. Note that the value γ_n above can depend on n. To obtain an explicit dependence, we simplify it using specific choices for η and β, as formalized below.

Theorem 2.3.2. *Suppose $f \in \mathcal{F}_n$. Let $\eta = \mu_0/(Ln^\alpha)$ ($0 < \mu_0 < 1$ and $0 < \alpha \leq 1$), $\beta = L/n^{\alpha/2}$, $m = \lfloor n^{3\alpha/2}/(3\mu_0) \rfloor$ and T is some multiple of m. Then there exists universal constants $\mu_0, \nu > 0$ such that we have the following: $\gamma_n \geq \frac{\nu}{Ln^\alpha}$ in Theorem 2.3.1 and*

$$\mathbb{E}[\|\nabla f(x_a)\|^2] \leq \frac{Ln^\alpha [f(x^0) - f(x^*)]}{T\nu},$$

where x^* is an optimal solution to the problem in (2.1) *and x_a is the output of Algorithm 2.*

By rewriting the above result in terms IFO calls, we get the following general corollary for nonconvex SVRG.

Corollary 2.3.2.1. *Suppose $f \in \mathcal{F}_n$. Then the IFO complexity of Algorithm 2 (with parameters from Theorem 2.3.2) for achieving an ϵ-accurate solution is:*

$$\text{IFO calls} = \begin{cases} O\left(n + (n^{1-\frac{\alpha}{2}}/\epsilon)\right), & \text{if } \alpha < 2/3, \\ O\left(n + (n^\alpha/\epsilon)\right), & \text{if } \alpha \geq 2/3. \end{cases}$$

Corollary 2.3.2.1 shows the interplay between step size and the IFO complexity. We observe that the number of IFO calls is minimized in Corollary 2.3.2.1 when $\alpha = 2/3$. This gives rise to the following key results of the chapter.

Corollary 2.3.2.2. *Suppose $f \in \mathcal{F}_n$. Let $\eta = \mu_1/(Ln^{2/3})$ ($0 < \mu_1 < 1$), $\beta = L/n^{1/3}$, $m = \lfloor n/(3\mu_1) \rfloor$ and T is some multiple of m. Then there exists universal constants $\mu_1, \nu_1 > 0$*

Algorithm 3: GD-SVRG$\left(x^0, K, T, m, \{p_i\}_{i=0}^m, \{\eta_i\}_{i=0}^{m-1}\right)$

```
1: Input: $x^0 \in \mathbb{R}^d$, $K$, epoch length $m$, step sizes $\{\eta_i > 0\}_{i=0}^{m-1}$, discrete probability distribution $\{p_i\}_{i=0}^m$
2: for $k = 0$ to $K$ do
3:     $x^k = \text{SVRG}(x^{k-1}, T, m, \{p_i\}_{i=0}^m, \{\eta_i\}_{i=0}^{m-1})$
4: end for
5: Output: $x^K$
```

such that we have the following: $\gamma_n \geq \frac{\nu_1}{Ln^{2/3}}$ *in Theorem 2.3.1 and*

$$\mathbb{E}[\|\nabla f(x_a)\|^2] \leq \frac{Ln^{2/3}[f(x^0) - f(x^*)]}{T\nu_1},$$

where x^* *is an optimal solution to the problem in* (2.1) *and* x_a *is the output of Algorithm 2.*

Corollary 2.3.2.3. *If* $f \in \mathcal{F}_n$, *then the IFO complexity of Algorithm 2 (with parameters in Corollary 2.3.2.2) to obtain an* ϵ*-accurate solution is* $O(n + (n^{2/3}/\epsilon))$.

Note the rate of $O(1/T)$ in the above results, as opposed to slower $O(1/\sqrt{T})$ rate of SGD (Theorem 2.2.3). For a more comprehensive comparison of the rates, refer to Section 2.6.

2.3.1 Gradient Dominated Functions

Before ending our discussion on convergence of nonconvex SVRG, we prove a linear convergence rate for the class of τ-gradient dominated functions (2.2). For ease of exposition, assume that $\tau > n^{1/3}$, a property analogous to the "high condition number regime" for strongly convex functions typical in machine learning. Note that gradient dominated functions can be nonconvex.

Theorem 2.3.3. *Suppose* f *is* τ*-gradient dominated where* $\tau > n^{1/3}$. *Then, the iterates of Algorithm 3 with* $T = \lceil 2L\tau n^{2/3}/\nu_1 \rceil$, $m = \lfloor n/(3\mu_1) \rfloor$, $\eta_t = \mu_1/(Ln^{2/3})$ *for all* $0 \leq t \leq m-1$ *and* $p_m = 1$ *and* $p_i = 0$ *for all* $0 \leq i < m$ *satisfy*

$$\mathbb{E}[\|\nabla f(x^k)\|^2] \leq 2^{-k}[\|\nabla f(x^0)\|^2].$$

Here μ_1 *and* ν_1 *are the constants used in Corollary 2.3.2.2.*

In fact, for τ-gradient dominated functions we can prove a stronger result of *global* linear convergence.

Theorem 2.3.4. *If* f *is* τ*-gradient dominated* ($\tau > n^{1/3}$), *then with* $T = \lceil 2L\tau n^{2/3}/\nu_1 \rceil$, $m = \lfloor n/(3\mu_1) \rfloor$, $\eta_t = \mu_1/(Ln^{2/3})$ *for* $0 \leq t \leq m-1$ *and* $p_m = 1$ *and* $p_i = 0$ *for all* $0 \leq i < m$, *the iterates of Algorithm 3 satisfy*

$$\mathbb{E}[f(x^k) - f(x^*)] \leq 2^{-k}[f(x^0) - f(x^*)].$$

Here μ_1, ν_1 *are as in Corollary 2.3.2.2;* x^* *is an optimal solution.*

An immediate consequence is the following.

Corollary 2.3.4.1. *If f is τ-gradient dominated, the IFO complexity of Algorithm 3 (with parameters from Theorem 2.3.3) to compute an ϵ-accurate solution is $O((n + \tau n^{2/3}) \log(1/\epsilon))$.*

Note that GD can also achieve linear convergence rate for gradient dominated functions [122]. However, GD requires $O(n + n\tau \log(1/\epsilon))$ IFO calls to obtain an ϵ-accurate solution as opposed to $O(n + n^{2/3}\tau \log(1/\epsilon))$ for SVRG. Similar (but not the same) gains can be seen for SVRG for strongly convex functions [71]. Also notice that we did not assume anything except smoothness on the *individual* functions f_i in the above results. In particular, the following corollary is also an immediate consequence.

Corollary 2.3.4.2. *If f is λ-strongly convex and the functions $\{f_i\}_{i=1}^n$ are possibly nonconvex, then the number of IFO calls made by Algorithm 3 (with parameters from Theorem 2.3.3) to compute an ϵ-accurate solution is $O((n + n^{2/3}\kappa) \log(1/\epsilon))$.*

Recall that here κ denotes the condition number L/λ for a λ-strongly convex function. Corollary 2.3.4.2 follows from Corollary 2.3.4.1 upon noting that λ-strongly convex function is $1/2\lambda$-gradient dominated. Theorem 2.3.4 generalizes the linear convergence result in [71] since it allows nonconvex f_i. Observe that Corollary 2.3.4.2 also applies when f_i is strongly convex for all $i \in [n]$, though in this case a more refined result can be proved [71].

Finally, we note that our result also improves on a recent result on SDCA in the setting of Corollary 2.3.4.2 when the condition number κ is reasonably large – a case that typically arises in machine learning. More precisely, for l_2-regularized empirical loss minimization, Shalev-Shwartz [154] show that SDCA requires $O((n + \kappa^2) \log(1/\epsilon)$ iterations when the f_i's are possibly nonconvex but their sum f is strongly convex. In comparison, we show that Algorithm 3 requires $O((n + n^{2/3}\kappa) \log(1/\epsilon))$ iterations, which is an improvement over SDCA when $\kappa > n^{2/3}$.

2.4 Convex Case

In the previous section, we showed nonconvex SVRG converges to a stationary point at the rate $O(n^{2/3}/T)$. A natural question is whether this rate can be improved if we assume convexity? We provide an affirmative answer. For non-strongly convex functions, this yields a *direct* analysis (i.e., not based on strongly convex perturbations) for SVRG. While we state our results in terms of stationarity gap $\|\nabla f(x)\|^2$ for the ease of comparison, our analysis also provides rates with respect to the optimality gap $[f(x) - f(x^*)]$ (see the proof of Theorem 2.4.1 in the appendix).

Theorem 2.4.1. *If f_i is convex for all $i \in [n]$, $p_i = 1/m$ for $0 \leq i \leq m-1$, and $p_m = 0$, then for Algorithm 2, we have*

$$\mathbb{E}[\|\nabla f(x_a)\|^2] \leq \frac{L\|x^0 - x^*\|^2 + 4mL^2\eta^2[f(x^0) - f(x^*)]}{T\eta(1 - 4L\eta)},$$

where x^* is optimal for (2.1) *and x_a is the output of Algorithm 2.*

We now state corollaries of this theorem that explicitly show the dependence on n in the convergence rates.

Corollary 2.4.1.1. *If $m = n$ and $\eta = 1/(8L\sqrt{n})$ in Theorem 2.4.1, then we have the following bound:*

$$\mathbb{E}[\|\nabla f(x_a)\|^2] \leq \frac{L\sqrt{n}(16L\|x^0 - x^*\|^2 + [f(x^0) - f(x^*)])}{T},$$

where x^* is optimal for (2.1) *and x_a is the output of Algorithm 2.*

The above result uses a step size that depends on n. For the convex case, we can also use step sizes independent of n. The following corollary states the associated result.

Corollary 2.4.1.2. *If $m = n$ and $\eta = 1/(8L)$ in Theorem 2.4.1, then we have the following bound:*

$$\mathbb{E}[\|\nabla f(x_a)\|^2] \leq \frac{L(16L\|x^0 - x^*\|^2 + n[f(x^0) - f(x^*)])}{T},$$

where x^* is optimal for (2.1) *and x_a is the output of Algorithm 2.*

We can rewrite these corollaries in terms of IFO complexity to get the following corollaries.

Corollary 2.4.1.3. *If f_i is convex for all $i \in [n]$, then the IFO complexity of Algorithm 2 (with parameters from Corollary 2.4.1.1) to compute an ϵ-accurate solution is $O(n + (\sqrt{n}/\epsilon))$.*

Corollary 2.4.1.4. *If f_i is convex for all $i \in [n]$, then the IFO complexity of Algorithm 2 (with parameters from Corollary 2.4.1.2) to compute ϵ-accurate solution is $O(n/\epsilon)$.*

These results follow from Corollary 2.4.1.1 and Corollary 2.4.1.2 and noting that for $m = O(n)$ the total IFO calls made by Algorithm 2 is $O(n)$. It is instructive to quantitatively compare Corollary 2.4.1.3 and Corollary2.4.1.4. With a step size independent of n, the convergence rate of SVRG has a dependence that is in the order of n (Corollary 2.4.1.2). But this dependence can be reduced to $\sqrt{n}$ by either carefully selecting a step size that diminishes with n (Corollary 2.4.1.1) or by using a good initial point x^0 obtained by, say, running $O(n)$ iterations of SGD.

We emphasize that the convergence rate for convex case can be improved significantly by slightly modifying the algorithm (either by adding an appropriate strongly convex perturbation [173] or by using a choice of m that changes with epoch [179]). However, it is not clear if these strategies provide any theoretical gains for the general nonconvex case.

2.5 Mini-batch Nonconvex SVRG

In this section, we study the mini-batch version of Algorithm 2. Mini-batching is a popular strategy, especially in multicore and distributed settings as it greatly helps one exploit parallelism and reduce the communication costs. The pseudocode for mini-batch nonconvex SVRG (Algorithm 4) is provided in the appendix. The key difference between the mini-batch SVRG and Algorithm 2 lies in lines 6 to 8. To use mini-batches we replace line 6 with sampling (with replacement) a mini-batch $I_t \subset [n]$ of size b; lines 7 to 8 are replaced with the following updates:

$$u_t^{s+1} = \tfrac{1}{|I_t|} \textstyle\sum_{i_t \in I_t} \left(\nabla f_{i_t}(x_t^{s+1}) - \nabla f_{i_t}(\tilde{x}^s) \right) + g^{s+1},$$
$$x_{t+1}^{s+1} = x_t^{s+1} - \eta_t u_t^{s+1}$$

When $b = 1$, this reduces to Algorithm 2. Mini-batch is typically used to reduce the variance of the stochastic gradient and increase the parallelism. Lemma 2.16.2 (in Section 2.16 of the appendix) shows the reduction in the variance of stochastic gradients with mini-batch size b. Using this lemma, one can derive the mini-batch equivalents of Lemma 2.11.1, Theorem 2.3.1 and Theorem 2.3.2. However, for the sake of brevity, we directly state the following main result for mini-batch SVRG.

Theorem 2.5.1. *Let $\overline{\gamma}_n$ denote the following quantity:*

$$\overline{\gamma}_n := \min_{0 \le t \le m-1} \quad \left(\eta - \tfrac{\bar{c}_{t+1}\eta}{\beta} - \eta^2 L - 2\bar{c}_{t+1}\eta^2\right).$$

where $\bar{c}_m = 0$, $\bar{c}_t = \bar{c}_{t+1}(1 + \eta\beta + {}^{2\eta^2 L^2}/_{b}) + {}^{\eta_t^2 L^3}/_{b}$ for $0 \le t \le m-1$. Suppose $\eta = \mu_2 b/(Ln^{2/3})$ ($0 < \mu_2 < 1$), $\beta = L/n^{1/3}$, $m = \lfloor n/(3b\mu_2) \rfloor$ and T is some multiple of m. Then for the mini-batch version of Algorithm 2 with mini-batch size $b < n^{2/3}$, there exists universal constants $\mu_2, \nu_2 > 0$ such that we have the following: $\overline{\gamma}_n \ge \frac{\nu_2 b}{Ln^{2/3}}$ and

$$\mathbb{E}[\|\nabla f(x_a)\|^2] \le \frac{Ln^{2/3}[f(x^0) - f(x^*)]}{bT\nu_2},$$

where x^* is optimal for (2.1).

It is important to compare this result with mini-batched SGD. For a batch size of b, SGD obtains a rate of $O(1/\sqrt{bT} + 1/T)$ [35] (obtainable by a simple modification of Theorem 2.2.3). Specifically, SGD has a $1/\sqrt{b}$ dependence on the batch size. In contrast, Theorem 2.5.1 shows that SVRG has a much better dependence of $1/b$ on the batch size. Hence, compared to SGD, SVRG allows more efficient mini-batching. More formally, in terms of IFO queries we have the following result.

Corollary 2.5.1.1. *If $f \in \mathcal{F}_n$, then the IFO complexity of the mini-batch version of Algorithm 2 (with parameters from Theorem 2.5.1 and mini-batch size $b < n^{2/3}$) to obtain an ϵ-accurate solution is $O(n + (n^{2/3}/\epsilon))$.*

Corollary 2.5.1.1 shows an interesting property of mini-batch SVRG. First, note that b IFO calls are required for calculating the gradient on a mini-batch of size b. Hence, SVRG does not gain on IFO complexity by using mini-batches. However, if the b gradients are calculated in parallel, then this leads to a theoretical linear speedup in multicore and distributed settings. In contrast, SGD does not yield an efficient mini-batch strategy [85].

2.6 Comparison of the Convergence Rates

In this section, we give a comprehensive comparison of results obtained in this chapter. In particular, we compare key aspects of the convergence rates for SGD, GD, and SVRG. The comparison is based on IFO complexity to achieve an ϵ-accurate solution.

Dependence on n: The number of IFO calls of SVRG and GD depend explicitly on n. In contrast, the number of oracle calls of SGD is independent of n (Theorem 2.2.3). However, this comes at the expense of worse dependence on ϵ. The number of IFO calls in GD is proportional to n. But for SVRG this dependence reduces to $n^{1/2}$ for convex (Corollary 2.4.1.1) and $n^{2/3}$ for nonconvex (Corollary 2.3.2.2) problems. Whether this difference in dependence on n is due to nonconvexity or just an artifact of our analysis is an interesting open problem.

Dependence on ϵ: The dependence on ϵ (or alternatively T) follows from the convergence rates of the algorithms. SGD is seen to depend as $O(1/\epsilon^2)$ on ϵ, regardless of convexity or nonconvexity. In contrast, for both convex and nonconvex settings, SVRG and GD converge as $O(1/\epsilon)$. Furthermore, for gradient dominated functions, SVRG and GD have global linear convergence. This speedup in convergence over SGD is especially significant when medium to high accuracy solutions are required (i.e., ϵ is small).

Assumptions used in analysis: It is important to understand the assumptions used in deriving the convergence rates. All algorithms assume Lipschitz continuous gradients. However, SGD requires two additional subtle but important assumptions: σ-bounded gradients and advance knowledge of T (since its step sizes depend on T). On the other hand, both SVRG and GD do not require these assumptions, and thus, are more flexible.

Step size / learning rates: It is valuable to compare the step sizes used by the algorithms. The step sizes of SGD shrink as the number of *iterations* T increases—an undesirable property. On the other hand, the step sizes of SVRG and GD are independent of T. Hence, both these algorithms can be executed with a fixed step size. However, SVRG uses step sizes that depend on n (see Corollary 2.3.2.2 and Corollary 2.4.1.1). A step size independent of n can be used for SVRG for convex f, albeit at cost of worse dependence on n (Corollary 2.4.1.2). GD does not have this issue as its step size is independent of both n and T.

Dependence on initial point and mini-batch: SVRG is more sensitive to the initial point in comparison to SGD. This can be seen by comparing Corollary 2.3.2.2 (of SVRG) to Theorem 2.2.3 (of SGD). Hence, it is important to use a good initial point for SVRG. Similarly, a good mini-batch can be beneficial to SVRG. Moreover, mini-batches not only provides parallelism but also good theoretical guarantees (see Theorem 2.5.1). In contrast, the performance gain in SGD with mini-batches is not very pronounced (see Section 2.5).

2.7 Best of Two Worlds

We have seen in the previous section that SVRG combines the benefits of both GD and SGD. We now show that these benefits of SVRG can be made more pronounced by an appropriate step size under additional assumptions. In this case, the IFO complexity of SVRG is lower than those of SGD and GD. This variant of SVRG (MSVRG) chooses a step size based on the total number of iterations T (or alternatively ϵ). For our discussion below, we assume that $T > n$.

Theorem 2.7.1. *Let $f \in \mathcal{F}_n$ have σ-bounded gradients. Let $\eta_t = \eta = \max\{c/\sqrt{T}, \mu_1/(Ln^{2/3})\}$ (μ_1 is the universal constant from Corollary 2.3.2.2), $m = \lfloor n/(3\mu_1) \rfloor$, and $c = \sqrt{\frac{f(x^0)-f(x^*)}{2L\sigma^2}}$. Further, let T be a multiple of m, $p_m = 1$, and $p_i = 0$ for $0 \leq i < m$. Then, the output x_a of Algorithm 2 satisfies*

$$\mathbb{E}[\|\nabla f(x_a)\|^2] \leq \bar{\nu} \min\left\{2\sqrt{\frac{2(f(x^0)-f(x^*))L}{T}}\sigma, \frac{Ln^{2/3}[f(x^0)-f(x^*)]}{T\nu_1}\right\},$$

where $\bar{\nu} > 0$ is a universal constant, ν_1 is the universal constant from Corollary 2.3.2.2 and x^* is an optimal solution to (2.1).

Corollary 2.7.1.1. *If $f \in \mathcal{F}_n$ has σ-bounded gradients, the IFO complexity of Algorithm 2 (with parameters from Theorem 2.7.1) to achieve an ϵ-accurate solution is $O(\min\{1/\epsilon^2, n^{2/3}/\epsilon\})$.*

An almost identical reasoning can be applied when f is convex to get the bounds specified in Table 2.1. Hence, we omit the details and directly state the following result.

Corollary 2.7.1.2. *Suppose f_i is convex for $i \in [n]$ and f has σ-bounded gradients, then the IFO complexity of Algorithm 2 (with step size $\eta = \max\{1/(L\sqrt{T}), 1/(8L\sqrt{n})\}$, $m = n$ and $p_i = 1/m$ for $0 \leq i \leq m-1$ and $p_m = 0$) to achieve an ϵ-accurate solution is $O(\min\{1/\epsilon^2, \sqrt{n}/\epsilon\})$.*

MSVRG has a convergence rate faster than those of both SGD and SVRG, though this benefit is not without cost. MSVRG, in contrast to SVRG, uses the additional assumption of σ-bounded gradients. Furthermore, its step size is *not* fixed since it depends on the number of iterations T. While it is often difficult in practice to compute the step size of MSVRG (Theorem 2.7.1), it is typical to try multiple step sizes and choose the one with the best results.

2.8 Experiments

We present our empirical results in this section. For our experiments, we study the problem of multiclass classification using neural networks. This is a typical nonconvex problem encountered in machine learning.

Experimental Setup. We train neural networks with one fully-connected hidden layer of 100 nodes and 10 softmax output nodes. We use ℓ_2-regularization for training. We use CIFAR-10[2], MNIST[3], and STL-10[4] datasets for our experiments. These datasets are standard in the neural networks literature. The ℓ_2 regularization is 1e-3 for CIFAR-10 and MNIST, and 1e-2 for STL-10. The features in the datasets are normalized to

[2]www.cs.toronto.edu/~kriz/cifar.html
[3]http://yann.lecun.com/exdb/mnist/
[4]https://cs.stanford.edu/~acoates/stl10/

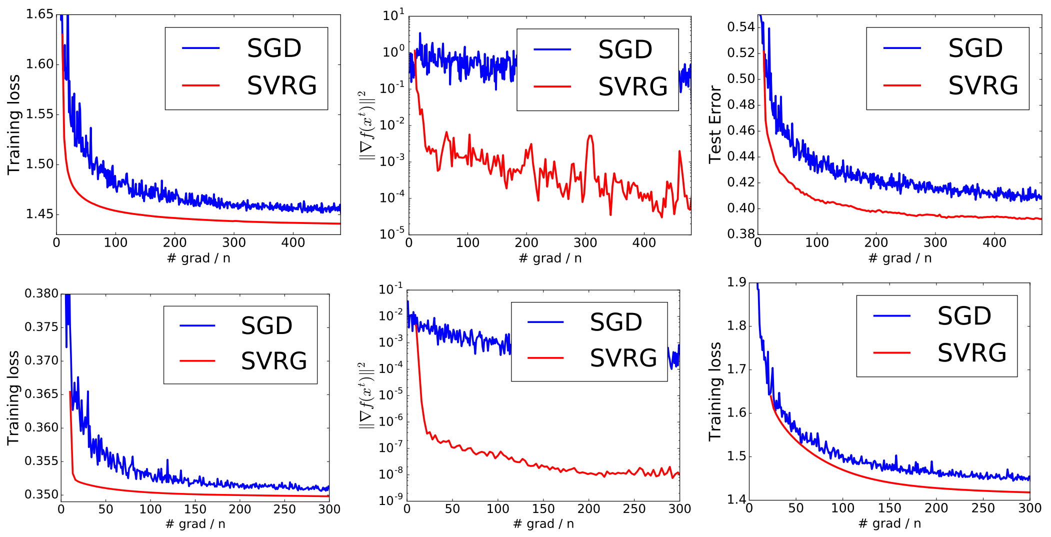


Figure 2.1: Neural network results for CIFAR-10, MNIST and STL-10 datasets. The top row represents the results for CIFAR-10 dataset. The bottom left and middle figures represent the results for MNIST dataset. The bottom right figure represents the result for STL-10.

the interval $[0, 1]$. All the datasets come with a predefined split into training and test datasets.

We compare SGD (the *de-facto* algorithm for training neural networks) against nonconvex SVRG. The step size (or learning rate) is critical for SGD. We set the learning rate of SGD using the popular $t-$inverse schedule $\eta_t = \eta_0(1 + \eta' \lfloor t/n \rfloor)^{-1}$, where η_0 and η' are chosen so that SGD gives the best performance on the training loss. In our experiments, we also use $\eta' = 0$; this results in a fixed step size for SGD. For SVRG, we use a fixed step size as suggested by our analysis. Again, the step size is chosen so that SVRG gives the best performance on the training loss.

Initialization & mini-batching. Initialization is critical to training of neural networks. We use the normalized initialization in [50] where parameters are chosen uniformly from $[-\sqrt{6/(n_i + n_o)}, \sqrt{6/(n_i + n_o)}]$ where n_i and n_o are the number of input and output layers of the neural network, respectively.

For SVRG, we use n iterations of SGD for CIFAR-10 and MINST and $2n$ iterations of SGD for STL-10 before running Algorithm 2. Such initialization is standard for variance reduced schemes even for convex problems [71, 153]. As noted earlier in Section 2.6, SVRG is more sensitive than SGD to the initial point, so such an initialization is typically helpful. We use mini-batches of size 10 in our experiments. SGD with mini-batches is common in training neural networks. Note that mini-batch training is especially beneficial for SVRG, as shown by our analysis in Section 2.5. Along the lines of theoretical analysis provided by Theorem 2.5.1, we use an epoch size $m = n/10$ in our experiments.

Results. We report objective function (training loss), test error (classification error on the test set), and $\|\nabla f(x^t)\|^2$ (convergence criterion throughout our analysis) for the datasets. For all the algorithms, we compare these criteria against the number of *effective passes* through the data, i.e., IFO calls divided by n. This includes the cost of calculating the full gradient at the end of each epoch of SVRG. Due to the SGD initialization in SVRG and mini-batching, the SVRG plots start from x-axis value of 10 for CIFAR-10 and MNIST and 20 for STL-10. Figure 2.1 shows the results for our experiment. It can be seen that the $\|\nabla f(x^t)\|^2$ for SVRG is lower compared to SGD, suggesting faster convergence to a stationary point. Furthermore, the training loss is also lower compared to SGD in all the datasets. Notably, the test error for CIFAR-10 is lower for SVRG, indicating better generalization; we did not notice substantial difference in test error for MNIST and STL-10 (see Section 2.17 in the appendix). Overall, these results on a network with one hidden layer are promising; it will be interesting to study SVRG for deep neural networks in the future.

2.9 Discussion

Before concluding the chapter, we would like to discuss the implications of our work and few caveats. One should exercise some caution while interpreting the results in the chapter. All our theoretical results are based on the stationarity gap. In general, this does not necessarily translate to optimality gap or low training loss and test error. One criticism against VR schemes in nonconvex optimization is the general wisdom that variance in the stochastic gradients of SGD can actually help it escape local minimum and saddle points. In fact, Ge et al. [46] add additional noise to the stochastic gradient in order to escape saddle points. However, one can reap the benefit of VR schemes even in such scenarios. For example, one can envision an algorithm which uses SGD as an exploration tool to obtain a good initial point and then uses a VR algorithm as an exploitation tool to quickly converge to a *good* local minimum. In either case, we believe variance reduction can be used as an important tool alongside other tools like momentum, adaptive learning rates for faster and better nonconvex optimization.

Appendix: Omitted Proofs and Additional Experiments

2.10 Nonconvex SGD: Convergence Rate

Proof of Theorem 2.2.3

Theorem. *Suppose f has σ-bounded gradient; let $\eta_t = \eta = c/\sqrt{T}$ where $c = \sqrt{\frac{2(f(x^0)-f(x^*))}{L\sigma^2}}$, and x^* is an optimal solution to* (2.1). *Then, the iterates of Algorithm 1 satisfy*

$$\min_{0\le t\le T-1} \mathbb{E}[\|\nabla f(x^t)\|^2] \le \sqrt{\frac{2(f(x^0)-f(x^*))L}{T}}\sigma.$$

Proof. We include the proof here for completeness. Please refer to [47] for a more general result.

The iterates of Algorithm 1 satisfy the following bound:

$$\mathbb{E}[f(x^{t+1})] \leq \mathbb{E}[f(x^t) + \left\langle \nabla f(x^t), x^{t+1} - x^t \right\rangle + \tfrac{L}{2}\|x^{t+1} - x^t\|^2] \tag{2.4}$$
$$\leq \mathbb{E}[f(x^t)] - \eta_t \mathbb{E}[\|\nabla f(x^t)\|^2] + \tfrac{L\eta_t^2}{2}\mathbb{E}[\|\nabla f_{i_t}(x^t)\|^2]$$
$$\leq \mathbb{E}[f(x^t)] - \eta_t \mathbb{E}[\|\nabla f(x^t)\|^2] + \tfrac{L\eta_t^2}{2}\sigma^2. \tag{2.5}$$

The first inequality follows from Lipschitz continuity of ∇f. The second inequality follows from the update in Algorithm 1 and since $\mathbb{E}_{i_t}[\nabla f_{i_t}(x^t)] = \nabla f(x^t)$ (unbiasedness of the stochastic gradient). The last step uses our assumption on gradient boundedness. Rearranging Equation (2.5) we obtain

$$\mathbb{E}[\|\nabla f(x^t)\|^2] \leq \tfrac{1}{\eta_t}\mathbb{E}[f(x^t) - f(x^{t+1})] + \tfrac{L\eta_t}{2}\sigma^2. \tag{2.6}$$

Summing Equation (2.6) from $t = 0$ to $T-1$ and using that η_t is constant η we obtain

$$\begin{aligned}\min_t \mathbb{E}[\|\nabla f(x^t)\|^2] &\leq \tfrac{1}{T}\textstyle\sum_{t=0}^{T-1} \mathbb{E}[\|f(x^t)\|^2] \\ &\leq \tfrac{1}{T\eta}\mathbb{E}[f(x^0) - f(x^T)] + \tfrac{L\eta}{2}\sigma^2 \\ &\leq \tfrac{1}{T\eta}(f(x^0) - f(x^*)) + \tfrac{L\eta}{2}\sigma^2 \\ &\leq \tfrac{1}{\sqrt{T}}\left(\tfrac{1}{c}(f(x^0) - f(x^*)) + \tfrac{Lc}{2}\sigma^2\right).\end{aligned}$$

The first step holds because the minimum is less than the average. The second and third steps are obtained from Equation (2.6) and the fact that $f(x^*) \leq f(x^T)$, respectively. The final inequality follows upon using $\eta = c/\sqrt{T}$. By setting

$$c = \sqrt{\frac{2(f(x^0) - f(x^*))}{L\sigma^2}}$$

in the above inequality, we get the desired result. □

2.11 Nonconvex SVRG

In this section, we provide the proofs of the results for nonconvex SVRG. We first start with few useful lemmas and then proceed towards the main results.

Lemma 2.11.1. *For $c_t, c_{t+1}, \beta_t > 0$, suppose we have*

$$c_t = c_{t+1}(1 + \eta_t\beta_t + 2\eta_t^2 L^2) + \eta_t^2 L^3.$$

Let η_t, β_t and c_{t+1} be chosen such that $\Gamma_t > 0$ (in Equation (2.3)). The iterate x_t^{s+1} in Algorithm 2 satisfy the bound:

$$\mathbb{E}[\|\nabla f(x_t^{s+1})\|^2] \leq \frac{R_t^{s+1} - R_{t+1}^{s+1}}{\Gamma_t},$$

where $R_t^{s+1} := \mathbb{E}[f(x_t^{s+1}) + c_t\|x_t^{s+1} - \tilde{x}^s\|^2]$ for $0 \leq s \leq S-1$.

Proof. Since f is L-smooth we have

$$\mathbb{E}[f(x_{t+1}^{s+1})] \leq \mathbb{E}[f(x_t^{s+1}) + \langle \nabla f(x_t^{s+1}), x_{t+1}^{s+1} - x_t^{s+1}\rangle + \tfrac{L}{2}\|x_{t+1}^{s+1} - x_t^{s+1}\|^2].$$

Using the SVRG update in Algorithm 2 and its unbiasedness, the right hand side above is further upper bounded by

$$\mathbb{E}[f(x_t^{s+1}) - \eta_t\|\nabla f(x_t^{s+1})\|^2 + \tfrac{L\eta_t^2}{2}\|v_t^{s+1}\|^2]. \tag{2.7}$$

Consider now the Lyapunov function

$$R_t^{s+1} := \mathbb{E}[f(x_t^{s+1}) + c_t\|x_t^{s+1} - \tilde{x}^s\|^2].$$

For bounding it we will require the following:

$$\begin{aligned}
&\mathbb{E}[\|x_{t+1}^{s+1} - \tilde{x}^s\|^2] = \mathbb{E}[\|x_{t+1}^{s+1} - x_t^{s+1} + x_t^{s+1} - \tilde{x}^s\|^2] \\
&= \mathbb{E}[\|x_{t+1}^{s+1} - x_t^{s+1}\|^2 + \|x_t^{s+1} - \tilde{x}^s\|^2 + 2\langle x_{t+1}^{s+1} - x_t^{s+1}, x_t^{s+1} - \tilde{x}^s\rangle] \\
&= \mathbb{E}[\eta_t^2\|v_t^{s+1}\|^2 + \|x_t^{s+1} - \tilde{x}^s\|^2] - 2\eta_t\mathbb{E}[\langle \nabla f(x_t^{s+1}), x_t^{s+1} - \tilde{x}^s\rangle] && (2.8)\\
&\leq \mathbb{E}[\eta_t^2\|v_t^{s+1}\|^2 + \|x_t^{s+1} - \tilde{x}^s\|^2] + 2\eta_t\mathbb{E}\left[\tfrac{1}{2\beta_t}\|\nabla f(x_t^{s+1})\|^2 + \tfrac{1}{2}\beta_t\|x_t^{s+1} - \tilde{x}^s\|^2\right]. && (2.9)
\end{aligned}$$

The second equality follows from the unbiasedness of the update of SVRG. The last inequality follows from a simple application of Cauchy-Schwarz and Young's inequality. Plugging Equation (2.7) and Equation (2.9) into R_{t+1}^{s+1}, we obtain the following bound:

$$\begin{aligned}
R_{t+1}^{s+1} &\leq \mathbb{E}[f(x_t^{s+1}) - \eta_t\|\nabla f(x_t^{s+1})\|^2 + \tfrac{L\eta_t^2}{2}\|v_t^{s+1}\|^2] \\
&\quad + \mathbb{E}[c_{t+1}\eta_t^2\|v_t^{s+1}\|^2 + c_{t+1}\|x_t^{s+1} - \tilde{x}^s\|^2] \\
&\quad + 2c_{t+1}\eta_t\mathbb{E}\left[\tfrac{1}{2\beta_t}\|\nabla f(x_t^{s+1})\|^2 + \tfrac{1}{2}\beta_t\|x_t^{s+1} - \tilde{x}^s\|^2\right] \\
&\leq \mathbb{E}[f(x_t^{s+1}) - \left(\eta_t - \tfrac{c_{t+1}\eta_t}{\beta_t}\right)\|\nabla f(x_t^{s+1})\|^2 \\
&\quad + \left(\tfrac{L\eta_t^2}{2} + c_{t+1}\eta_t^2\right)\mathbb{E}[\|v_t^{s+1}\|^2] \\
&\quad + (c_{t+1} + c_{t+1}\eta_t\beta_t)\,\mathbb{E}\left[\|x_t^{s+1} - \tilde{x}^s\|^2\right].
\end{aligned} \tag{2.10}$$

To further bound this quantity, we use Lemma 2.16.1 to bound $\mathbb{E}[\|v_t^{s+1}\|^2]$, so that upon substituting it in Equation (2.10), we see that

$$\begin{aligned} R_{t+1}^{s+1} &\leq \mathbb{E}[f(x_t^{s+1})] \\ &\quad - \left(\eta_t - \tfrac{c_{t+1}\eta_t}{\beta_t} - \eta_t^2 L - 2c_{t+1}\eta_t^2\right) \mathbb{E}[\|\nabla f(x_t^{s+1})\|^2] \\ &\quad + \left[c_{t+1}(1 + \eta_t\beta_t + 2\eta_t^2 L^2) + \eta_t^2 L^3\right] \mathbb{E}\left[\|x_t^{s+1} - \tilde{x}^s\|^2\right] \\ &\leq R_t^{s+1} - \left(\eta_t - \tfrac{c_{t+1}\eta_t}{\beta_t} - \eta_t^2 L - 2c_{t+1}\eta_t^2\right)\mathbb{E}[\|\nabla f(x_t^{s+1})\|^2]. \end{aligned}$$

The second inequality follows from the definition of c_t and R_t^{s+1}, thus concluding the proof. □

Proof of Theorem 2.3.1

Theorem. *Let $f \in \mathcal{F}_n$. Let $c_m = 0$, $\eta_t = \eta > 0$, $\beta_t = \beta > 0$, and $c_t = c_{t+1}(1 + \eta\beta + 2\eta^2 L^2) + \eta^2 L^3$ such that $\Gamma_t > 0$ for $0 \leq t \leq m-1$. Define the quantity $\gamma_n := \min_t \Gamma_t$. Further, let $p_i = 0$ for $0 \leq i < m$ and $p_m = 1$, and let T be a multiple of m. Then for the output x_a of Algorithm 2 we have*

$$\mathbb{E}[\|\nabla f(x_a)\|^2] \leq \frac{f(x^0) - f(x^*)}{T\gamma_n},$$

where x^* is an optimal solution to (2.1).

Proof. Since $\eta_t = \eta$ for $t \in \{0, \dots, m-1\}$, using Lemma 2.11.1 and telescoping the sum, we obtain

$$\textstyle\sum_{t=0}^{m-1} \mathbb{E}[\|\nabla f(x_t^{s+1})\|^2] \leq \frac{R_0^{s+1} - R_m^{s+1}}{\gamma_n}.$$

This inequality in turn implies that

$$\textstyle\sum_{t=0}^{m-1} \mathbb{E}[\|\nabla f(x_t^{s+1})\|^2] \leq \frac{\mathbb{E}[f(\tilde{x}^s) - f(\tilde{x}^{s+1})]}{\gamma_n}, \tag{2.11}$$

where we used that $R_m^{s+1} = \mathbb{E}[f(x_m^{s+1})] = \mathbb{E}[f(\tilde{x}^{s+1})]$ (since $c_m = 0$, $p_m = 1$, and $p_i = 0$ for $i < m$), and that $R_0^{s+1} = \mathbb{E}[f(\tilde{x}^s)]$ (since $x_0^{s+1} = \tilde{x}^s$, as $p_m = 1$ and $p_i = 0$ for $i < m$). Now sum over all epochs to obtain

$$\frac{1}{T}\sum_{s=0}^{S-1}\sum_{t=0}^{m-1} \mathbb{E}[\|\nabla f(x_t^{s+1})\|^2] \leq \frac{f(x^0) - f(x^*)}{T\gamma_n}. \tag{2.12}$$

The above inequality used the fact that $\tilde{x}^0 = x^0$. Using the above inequality and the definition of x_a in Algorithm 2, we obtain the desired result. □

Proof of Theorem 2.3.2

Theorem. *Suppose $f \in \mathcal{F}_n$. Let $\eta = \mu_0/(Ln^\alpha)$ ($0 < \mu_0 < 1$ and $0 < \alpha \leq 1$), $\beta = L/n^{\alpha/2}$, $m = \lfloor n^{3\alpha/2}/(3\mu_0) \rfloor$ and T is some multiple of m. Then there exists universal constants $\mu_0, \nu > 0$ such that we have the following: $\gamma_n \geq \frac{\nu}{Ln^\alpha}$ in Theorem 2.3.1 and*

$$\mathbb{E}[\|\nabla f(x_a)\|^2] \leq \frac{Ln^\alpha[f(x^0) - f(x^*)]}{T\nu},$$

where x^* is an optimal solution to the problem in (2.1) and x_a is the output of Algorithm 2.

Proof. For our analysis, we will require an upper bound on c_0. We observe that $c_0 = \frac{\mu_0^2 L}{n^{2\alpha}} \frac{(1+\theta)^m - 1}{\theta}$ where $\theta = 2\eta^2 L^2 + \eta\beta$. This is obtained using the relation $c_t = c_{t+1}(1 + \eta\beta + 2\eta^2 L^2) + \eta^2 L^3$ and the fact that $c_m = 0$. Using the specified values of β and η we have

$$\theta = 2\eta^2 L^2 + \eta\beta = \frac{2\mu_0^2}{n^{2\alpha}} + \frac{\mu_0}{n^{3\alpha/2}} \leq \frac{3\mu_0}{n^{3\alpha/2}}.$$

The above inequality follows since $\mu_0 \leq 1$ and $n \geq 1$. Using the above bound on θ, we get

$$\begin{aligned} c_0 = \frac{\mu_0^2 L}{n^{2\alpha}} \frac{(1+\theta)^m - 1}{\theta} &= \frac{\mu_0 L((1+\theta)^m - 1)}{2\mu_0 + n^{\alpha/2}} \\ &\leq \frac{\mu_0 L((1 + \frac{3\mu_0}{n^{3\alpha/2}})^{\lfloor n^{3\alpha/2}/3\mu_0 \rfloor} - 1)}{2\mu_0 + n^{\alpha/2}} \\ &\leq n^{-\alpha/2}(\mu_0 L(e-1)), \end{aligned} \tag{2.13}$$

wherein the second inequality follows upon noting that $(1 + \frac{1}{l})^l$ is increasing for $l > 0$ and $\lim_{l\to\infty}(1 + \frac{1}{l})^l = e$ (here e is the Euler's number). Now we can lower bound γ_n, as

$$\begin{aligned} \gamma_n &= \min_t \big(\eta - \tfrac{c_{t+1}\eta}{\beta} - \eta^2 L - 2c_{t+1}\eta^2\big) \\ &\geq \big(\eta - \tfrac{c_0\eta}{\beta} - \eta^2 L - 2c_0\eta^2\big) \geq \frac{\nu}{Ln^\alpha}, \end{aligned}$$

where ν is a constant independent of n. The first inequality holds since c_t decreases with t. The second inequality holds since (a) c_0/β is upper bounded by a constant independent of n as $c_0/\beta \leq \mu_0(e-1)$ (follows from Equation (2.13)), (b) $\eta^2 L \leq \mu_0\eta$ and (c) $2c_0\eta^2 \leq 2\mu_0^2(e-1)\eta$ (follows from Equation (2.13)). By choosing μ_0 (independent of n) appropriately, one can ensure that $\gamma_n \geq \nu/(Ln^\alpha)$ for some universal constant ν. For example, choosing $\mu_0 = 1/4$, we have $\gamma_n \geq \nu/(Ln^\alpha)$ with $\nu = 1/40$. Substituting the above lower bound in Equation (2.12), we obtain the desired result. $\square$

Proof of Corollary 2.3.2.1

Corollary. *Suppose $f \in \mathcal{F}_n$. Then the IFO complexity of Algorithm 2 (with parameters from Theorem 2.3.2) for achieving an ϵ-accurate solution is:*

$$\textit{IFO calls} = \begin{cases} O\left(n + (n^{1-\frac{\alpha}{2}}/\epsilon)\right), & \textit{if } \alpha < 2/3, \\ O\left(n + (n^{\alpha}/\epsilon)\right), & \textit{if } \alpha \geq 2/3. \end{cases}$$

Proof. This result follows from Theorem 2.3.2 and the fact that $m = \lfloor n^{3\alpha/2}/(3\mu_0) \rfloor$. Suppose $\alpha < 2/3$, then $m = o(n)$. However, n IFO calls are invested in calculating the average gradient at the end of each epoch. In other words, computation of average gradient requires n IFO calls for every m iterations of the algorithm. Using this relationship, we get $O\big(n + (n^{1-\frac{\alpha}{2}}/\epsilon)\big)$ in this case.

On the other hand, when $\alpha \geq 2/3$, the total number of IFO calls made by Algorithm 2 in each epoch is $\Omega(n)$ since $m = \lfloor n^{3\alpha/2}/(3\mu_0) \rfloor$. Hence, the oracle calls required for calculating the average gradient (per epoch) is of lower order, leading to $O\big(n + (n^{\alpha}/\epsilon)\big)$ IFO calls. □

2.12 GD-SVRG

Proof of Theorem 2.3.3

Theorem. *Suppose f is τ-gradient dominated where $\tau > n^{1/3}$. Then, the iterates of Algorithm 3 with $T = \lceil 2L\tau n^{2/3}/\nu_1 \rceil$, $m = \lfloor n/(3\mu_1) \rfloor$, $\eta_t = \mu_1/(Ln^{2/3})$ for all $0 \leq t \leq m-1$ and $p_m = 1$ and $p_i = 0$ for all $0 \leq i < m$ satisfy*

$$\mathbb{E}[\|\nabla f(x^k)\|^2] \leq 2^{-k}[\|\nabla f(x^0)\|^2].$$

Here μ_1 and ν_1 are the constants used in Corollary 2.3.2.2.

Proof. Corollary 2.3.2.2 shows that the iterates of Algorithm 3 satisfy

$$\mathbb{E}[\|\nabla f(x^k)\|^2] \leq \frac{Ln^{2/3}\mathbb{E}[f(x^{k-1}) - f(x^*)]}{T\nu_1}.$$

Substituting the specified value of T in the above inequality, we have

$$\begin{aligned} \mathbb{E}[\|\nabla f(x^k)\|^2] &\leq \frac{1}{2\tau}\big(\mathbb{E}[f(x^{k-1}) - f(x^*)]\big) \\ &\leq \tfrac{1}{2}\mathbb{E}[\|\nabla f(x^{k-1})\|^2]. \end{aligned}$$

The second inequality follows from τ-gradient dominance of the function f. □

Proof of Theorem 2.3.4

Theorem. *If f is τ-gradient dominated ($\tau > n^{1/3}$), then with $T = \lceil 2L\tau n^{2/3}/\nu_1 \rceil$, $m = \lfloor n/(3\mu_1) \rfloor$, $\eta_t = \mu_1/(Ln^{2/3})$ for $0 \leq t \leq m-1$ and $p_m = 1$ and $p_i = 0$ for all $0 \leq i < m$, the iterates of Algorithm 3 satisfy*

$$\mathbb{E}[f(x^k) - f(x^*)] \leq 2^{-k}[f(x^0) - f(x^*)].$$

Here μ_1, ν_1 are as in Corollary 2.3.2.2; x^* is an optimal solution.

Proof. The proof mimics that of Theorem 2.3.3; now we have the following condition on the iterates of Algorithm 3:

$$\mathbb{E}[\|\nabla f(x^k)\|^2] \leq \frac{\mathbb{E}[f(x^{k-1}) - f(x^*)]}{2\tau}. \tag{2.14}$$

However, f is τ-gradient dominated, so $\mathbb{E}[\|\nabla f(x^k)\|^2] \geq \mathbb{E}[f(x^k) - f(x^*)]/\tau$, which combined with Equation (2.14) concludes the proof. □

2.13 Convex SVRG: Convergence Rate

Proof of Theorem 2.4.1

Theorem. *If f_i is convex for all $i \in [n]$, $p_i = 1/m$ for $0 \leq i \leq m-1$, and $p_m = 0$, then for Algorithm 2, we have*

$$\mathbb{E}[\|\nabla f(x_a)\|^2] \leq \frac{L\|x^0 - x^*\|^2 + 4mL^2\eta^2[f(x^0) - f(x^*)]}{T\eta(1 - 4L\eta)},$$

where x^* is optimal for (2.1) *and x_a is the output of Algorithm 2.*

Proof. Consider the following sequence of inequalities:

$$\begin{aligned}
&\mathbb{E}[\|x_{t+1}^{s+1} - x^*\|^2] = \mathbb{E}[\|x_t^{s+1} - \eta v_t^{s+1} - x^*\|^2] \\
&\leq \mathbb{E}[\|x_t^{s+1} - x^*\|^2] + \eta^2\mathbb{E}[\|v_t^{s+1}\|^2] \\
&\qquad - 2\eta\mathbb{E}[\langle v_t^{s+1}, x_t^{s+1} - x^*\rangle] \\
&\leq \mathbb{E}[\|x_t^{s+1} - x^*\|^2] + \eta^2\mathbb{E}[\|v_t^{s+1}\|^2] \\
&\qquad - 2\eta\mathbb{E}[f(x_t^{s+1}) - f(x^*)] \\
&\leq \mathbb{E}[\|x_t^{s+1} - x^*\|^2] - 2\eta(1 - 2L\eta)\mathbb{E}[f(x_t^{s+1}) - f(x^*)] \\
&\qquad + 4L\eta^2\mathbb{E}[f(\tilde{x}^s) - f(x^*)] \\
&= \mathbb{E}[\|x_t^{s+1} - x^*\|^2] - 2\eta(1 - 4L\eta)\mathbb{E}[f(x_t^{s+1}) - f(x^*)] \\
&\qquad + 4L\eta^2\mathbb{E}[f(\tilde{x}^s) - f(x^*)] - 4L\eta^2\mathbb{E}[f(x_t^{s+1}) - f(x^*)].
\end{aligned}$$

The second inequality uses unbiasedness of the SVRG update and convexity of f. The third inequality follows from Lemma 2.18.2. Defining the Lyapunov function

$$P^s := \mathbb{E}[\|x^s_m - x^*\|^2] + 4mL\eta^2\mathbb{E}[f(\tilde{x}^s) - f(x^*)],$$

and summing the above inequality over t, we get

$$2\eta(1-4L\eta)\sum_{t=0}^{m-1}\mathbb{E}[f(x^{s+1}_t) - f(x^*)] \le P^s - P^{s+1}.$$

This due is to the fact that

$$\begin{aligned} P^{s+1} &= \mathbb{E}[\|x^{s+1}_m - x^*\|^2] + 4mL\eta^2\mathbb{E}[f(\tilde{x}^{s+1}) - f(x^*)] \\ &= \mathbb{E}[\|x^{s+1}_m - x^*\|^2] + 4L\eta^2\sum_{t=0}^{m-1}\mathbb{E}[f(x^{s+1}_t) - f(x^*)]. \end{aligned}$$

The above equality uses the fact that $p_m = 0$ and $p_i = 1/m$ for $0 \le i < m$. Summing over all epochs and telescoping we then obtain

$$\mathbb{E}[f(x_a) - f(x^*)] \le P^0\big(2T\eta(1-4L\eta)\big)^{-1}.$$

The inequality also uses the definition of x_a given in Alg 2. On this inequality we use Lemma 2.18.1, which yields

$$\begin{aligned} \mathbb{E}[\|\nabla f(x_a)\|^2] &\le 2L\mathbb{E}[f(x_a) - f(x^*)] \\ &\le \frac{L\|x^0 - x^*\|^2 + 4mL^2\eta^2[f(x^0) - f(x^*)]}{T\eta(1-4L\eta)}. \end{aligned} \qquad \square$$

It is easy to see that we can obtain convergence rates for $E[f(x_a) - f(x^*)]$ from the above reasoning. This leads to a *direct* analysis of SVRG for convex functions.

2.14 Minibatch Nonconvex SVRG

Proof of Theorem 2.5.1

The proofs essentially follow along the lines of Lemma 2.11.1, Theorem 2.3.1 and Theorem 2.3.2 with the added complexity of mini-batch. We first prove few intermediate results before proceeding to the proof of Theorem 2.5.1.

Lemma 2.14.1. *Suppose we have*

$$\begin{aligned} \overline{R}^{s+1}_t &:= \mathbb{E}[f(x^{s+1}_t) + \bar{c}_t\|x^{s+1}_t - \tilde{x}^s\|^2], \\ \bar{c}_t &= \bar{c}_{t+1}(1 + \eta_t\beta_t + \frac{2\eta_t^2L^2}{b}) + \frac{\eta_t^2L^3}{b}, \end{aligned}$$

Algorithm 4: Mini-batch SVRG

```
1: Input: $\tilde{x}^0 = x^0_m = x^0 \in \mathbb{R}^d$, epoch length $m$, step sizes $\{\eta_i > 0\}_{i=0}^{m-1}$, $S = \lceil T/m \rceil$, discrete
   probability distribution $\{p_i\}_{i=0}^{m}$, mini-batch size $b$
2: for $s = 0$ to $S-1$ do
3:   $x_0^{s+1} = x_m^s$
4:   $g^{s+1} = \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(\tilde{x}^s)$
5:   for $t = 0$ to $m-1$ do
6:     Choose a mini-batch (uniformly random with replacement) $I_t \subset [n]$ of size $b$
7:     $u_t^{s+1} = \frac{1}{b}\sum_{i_t \in I_t}(\nabla f_{i_t}(x_t^{s+1}) - \nabla f_{i_t}(\tilde{x}^s)) + g^{s+1}$
8:     $x_{t+1}^{s+1} = x_t^{s+1} - \eta_t u_t^{s+1}$
9:   end for
10:  $\tilde{x}^{s+1} = \sum_{i=0}^{m} p_i x_i^{s+1}$
11: end for
12: Output: Iterate $x_a$ chosen uniformly random from $\{\{x_t^{s+1}\}_{t=0}^{m-1}\}_{s=0}^{S-1}$.
```

for $0 \le s \le S-1$ *and* $0 \le t \le m-1$ *and the parameters* η_t, β_t *and* $\bar{c}_{t+1}$ *are chosen such that*

$$\left(\eta_t - \frac{\bar{c}_{t+1}\eta_t}{\beta_t} - \eta_t^2 L - 2\bar{c}_{t+1}\eta_t^2\right) \ge 0.$$

Then the iterates x_t^{s+1} *in the mini-batch version of Algorithm 2 i.e., Algorithm 4 with mini-batch size b satisfy the bound:*

$$\mathbb{E}[\|\nabla f(x_t^{s+1})\|^2] \le \frac{\overline{R}_t^{s+1} - \overline{R}_{t+1}^{s+1}}{\left(\eta_t - \frac{\bar{c}_{t+1}\eta_t}{\beta_t} - \eta_t^2 L - 2\bar{c}_{t+1}\eta_t^2\right)},$$

Proof. Using essentially the same argument as the proof of Lemma. 2.11.1 until Equation (2.10), we have

$$\begin{aligned} \overline{R}_{t+1}^{s+1} \le \mathbb{E}[f(x_t^{s+1}) &- \left(\eta_t - \tfrac{\bar{c}_{t+1}\eta_t}{\beta_t}\right)\|\nabla f(x_t^{s+1})\|^2 \\ &+ \left(\tfrac{L\eta_t^2}{2} + \bar{c}_{t+1}\eta_t^2\right)\mathbb{E}[\|u_t^{s+1}\|^2] \\ &+ (\bar{c}_{t+1} + \bar{c}_{t+1}\eta_t\beta_t)\,\mathbb{E}\left[\|x_t^{s+1} - \tilde{x}^s\|^2\right]. \end{aligned} \tag{2.15}$$

We use Lemma 2.16.2 in order to bound $\mathbb{E}[\|u_t^{s+1}\|^2]$ in the above inequality. Substituting it in Equation (2.15), we see that

$$\begin{aligned} \overline{R}_{t+1}^{s+1} &\le \mathbb{E}[f(x_t^{s+1})] \\ &\quad - \left(\eta_t - \tfrac{\bar{c}_{t+1}\eta_t}{\beta_t} - \eta_t^2 L - 2\bar{c}_{t+1}\eta_t^2\right)\mathbb{E}[\|\nabla f(x_t^{s+1})\|^2] \\ &\quad + \left[\bar{c}_{t+1}\left(1 + \eta_t\beta_t + \tfrac{2\eta_t^2 L^2}{b}\right) + \tfrac{\eta_t^2 L^3}{b}\right]\mathbb{E}\left[\|x_t^{s+1} - \tilde{x}^s\|^2\right] \\ &\le \overline{R}_t^{s+1} - \left(\eta_t - \tfrac{\bar{c}_{t+1}\eta_t}{\beta_t} - \eta_t^2 L - 2\bar{c}_{t+1}\eta_t^2\right)\mathbb{E}[\|\nabla f(x_t^{s+1})\|^2]. \end{aligned}$$

The second inequality follows from the definition of $\bar{c}_t$ and $\overline{R}_t^{s+1}$, thus concluding the proof. □

Our intermediate key result is the following theorem that provides convergence rate of mini-batch SVRG.

Theorem 2.14.2. *Let $\overline{\gamma}_n$ denote the following quantity:*

$$\overline{\gamma}_n := \min_{0 \le t \le m-1} \quad \left(\eta - \tfrac{\bar{c}_{t+1}\eta}{\beta} - \eta^2 L - 2\bar{c}_{t+1}\eta^2\right).$$

Suppose $\eta_t = \eta$ and $\beta_t = \beta$ for all $t \in \{0, \dots, m-1\}$, $\bar{c}_m = 0$, $\bar{c}_t = \bar{c}_{t+1}(1 + \eta_t\beta_t + \frac{2\eta_t^2 L^2}{b}) + \frac{\eta_t^2 L^3}{b}$ for $t \in \{0, \dots, m-1\}$ and $\overline{\gamma}_n > 0$. Further, let $p_m = 1$ and $p_i = 0$ for $0 \le i < m$. Then for the output x_a of mini-batch version of Algorithm 2 with mini-batch size b, we have

$$\mathbb{E}[\|\nabla f(x_a)\|^2] \le \frac{f(x^0) - f(x^*)}{T\overline{\gamma}_n},$$

where x^* is an optimal solution to (2.1).

Proof. Since $\eta_t = \eta$ for $t \in \{0, \dots, m-1\}$, using Lemma 2.14.1 and telescoping the sum, we obtain

$$\textstyle\sum_{t=0}^{m-1} \mathbb{E}[\|\nabla f(x_t^{s+1})\|^2] \le \frac{\overline{R}_0^{s+1} - \overline{R}_m^{s+1}}{\overline{\gamma}_n}.$$

This inequality in turn implies that

$$\textstyle\sum_{t=0}^{m-1} \mathbb{E}[\|\nabla f(x_t^{s+1})\|^2] \le \frac{\mathbb{E}[f(\tilde{x}^s) - f(\tilde{x}^{s+1})]}{\overline{\gamma}_n},$$

where we used that $\overline{R}_m^{s+1} = \mathbb{E}[f(x_m^{s+1})] = \mathbb{E}[f(\tilde{x}^{s+1})]$ (since $\bar{c}_m = 0$, $p_m = 1$, and $p_i = 0$ for $i < m$), and that $\overline{R}_0^{s+1} = \mathbb{E}[f(\tilde{x}^s)]$ (since $x_0^{s+1} = \tilde{x}^s$, as $p_m = 1$ and $p_i = 0$ for $i < m$). Now sum over all epochs and using the fact that $\tilde{x}^0 = x^0$, we get the desired result. □

We now present the proof of Theorem 2.5.1 using the above results.

Theorem. *Let $\overline{\gamma}_n$ denote the following quantity:*

$$\overline{\gamma}_n := \min_{0 \le t \le m-1} \quad \left(\eta - \tfrac{\bar{c}_{t+1}\eta}{\beta} - \eta^2 L - 2\bar{c}_{t+1}\eta^2\right).$$

where $\bar{c}_m = 0$, $\bar{c}_t = \bar{c}_{t+1}(1 + \eta\beta + {}^{2\eta^2 L^2}/_{b}) + {}^{\eta_t^2 L^3}/_{b}$ for $0 \le t \le m-1$. Suppose $\eta = \mu_2 b/(Ln^{2/3})$ $(0 < \mu_2 < 1)$, $\beta = L/n^{1/3}$, $m = \lfloor n/(3b\mu_2) \rfloor$ and T is some multiple of m. Then for the mini-batch version of Algorithm 2 with mini-batch size $b < n^{2/3}$, there exists universal constants $\mu_2, \nu_2 > 0$ such that we have the following: $\overline{\gamma}_n \ge \frac{\nu_2 b}{Ln^{2/3}}$ and

$$\mathbb{E}[\|\nabla f(x_a)\|^2] \le \frac{Ln^{2/3}[f(x^0) - f(x^*)]}{bT\nu_2},$$

where x^* is optimal for (2.1).

Proof of Theorem 2.5.1. We first observe that using the specified values of β and η we obtain

$$\overline{\theta} := \frac{2\eta^2 L^2}{b} + \eta\beta = \frac{2\mu_2^2 b}{n^{4/3}} + \frac{\mu_2 b}{n} \leq \frac{3\mu_2 b}{n}.$$

The above inequality follows since $\mu_2 \leq 1$ and $n \geq 1$. For our analysis, we will require the following bound on $\overline{c}_0$:

$$\begin{aligned}\overline{c}_0 &= \frac{\mu_2^2 b^2 L}{bn^{4/3}} \frac{(1+\overline{\theta})^m - 1}{\overline{\theta}} = \frac{\mu_2 bL((1+\overline{\theta})^m - 1)}{2b\mu_2 + bn^{1/3}} \\ &\leq n^{-1/3}(\mu_2 L(e-1)),\end{aligned} \tag{2.16}$$

wherein the first equality holds due to the relation $\overline{c}_t = \overline{c}_{t+1}(1 + \eta_t\beta_t + \frac{2\eta_t^2 L^2}{b}) + \frac{\eta_t^2 L^3}{b}$, and the inequality follows upon again noting that $(1+1/l)^l$ is increasing for $l > 0$ and $\lim_{l\to\infty}(1+\frac{1}{l})^l = e$. Now we can lower bound $\overline{\gamma}_n$, as

$$\begin{aligned}\overline{\gamma}_n &= \min_t \big(\eta - \tfrac{\overline{c}_{t+1}\eta}{\beta} - \eta^2 L - 2\overline{c}_{t+1}\eta^2\big) \\ &\geq \big(\eta - \tfrac{\overline{c}_0\eta}{\beta} - \eta^2 L - 2\overline{c}_0\eta^2\big) \geq \frac{b\nu_2}{Ln^{2/3}},\end{aligned}$$

where ν_2 is a constant independent of n. The first inequality holds since $\overline{c}_t$ decreases with t. The second one holds since (a) $\overline{c}_0/\beta$ is upper bounded by a constant independent of n as $\overline{c}_0/\beta \leq \mu_2(e-1)$ (due to Equation (2.16)), (b) $\eta^2 L \leq \mu_2\eta$ (as $b < n^{2/3}$) and (c) $2\overline{c}_0\eta^2 \leq 2\mu_2^2(e-1)\eta$ (again due to Equation (2.16) and the fact $b < n^{2/3}$). By choosing an appropriately small constant μ_2 (independent of n), one can ensure that $\overline{\gamma}_n \geq b\nu_2/(Ln^{2/3})$ for some universal constant ν_2. For example, choosing $\mu_2 = 1/4$, we have $\overline{\gamma}_n \geq b\nu_2/(Ln^{2/3})$ with $\nu_2 = 1/40$. Substituting the above lower bound in Theorem 2.14.2, we get the desired result. □

2.15 MSVRG: Convergence Rate

Proof of Theorem 2.7.1

Theorem. *Let $f \in \mathcal{F}_n$ have σ-bounded gradients. Let $\eta_t = \eta = \max\{c/\sqrt{T}, \mu_1/(Ln^{2/3})\}$ (μ_1 is the universal constant from Corollary 2.3.2.2), $m = \lfloor n/(3\mu_1) \rfloor$, and $c = \sqrt{\frac{f(x^0)-f(x^*)}{2L\sigma^2}}$. Further, let T be a multiple of m, $p_m = 1$, and $p_i = 0$ for $0 \leq i < m$. Then, the output x_a of Algorithm 2 satisfies*

$$\begin{aligned}&\mathbb{E}[\|\nabla f(x_a)\|^2] \\ &\leq \bar{\nu} \min\Big\{2\sqrt{\frac{2(f(x^0) - f(x^*))L}{T}}\sigma, \frac{Ln^{2/3}[f(x^0) - f(x^*)]}{T\nu_1}\Big\},\end{aligned}$$

where $\bar{\nu}$ is a universal constant, ν_1 is the universal constant from Corollary 2.3.2.2 and x^* is an optimal solution to (2.1).

Proof. First, we observe that the step size η is chosen to be $\max\{c/\sqrt{T}, \mu_1/(Ln^{2/3})\}$ where

$$c = \sqrt{\frac{f(x^0) - f(x^*)}{2L\sigma^2}}.$$

Suppose $\eta = \mu_1/(Ln^{2/3})$, we obtain the convergence rate in Corollary 2.3.2.2. Now, lets consider the case where $\eta = c/\sqrt{T}$. In this case, we have the following bound:

$$\begin{aligned}
&\mathbb{E}[\|v_t^{s+1}\|^2] \\
&= \mathbb{E}[\|\nabla f_{i_t}(x_t^{s+1}) - \nabla f_{i_t}(\tilde{x}^s) + \nabla f(\tilde{x}^s)\|^2] \\
&\leq 2\left(\mathbb{E}[\|\nabla f_{i_t}(x_t^{s+1})\|^2 + \|\nabla f_{i_t}(\tilde{x}^s) - \nabla f(\tilde{x}^s)\|^2]\right) \\
&\leq 2\left(\mathbb{E}[\|\nabla f_{i_t}(x_t^{s+1})\|^2 + \|\nabla f_{i_t}(\tilde{x}^s)\|^2]\right) \\
&\leq 4\sigma^2.
\end{aligned}$$

The first inequality follows from Lemma 2.18.4 with $r = 2$. The second inequality follows from (a) σ-bounded gradient property of f and (b) the fact that for a random variable ζ, $\mathbb{E}[\|\zeta - \mathbb{E}[\zeta]\|^2] \leq \mathbb{E}[\|\zeta\|^2]$. The rest of the proof is along exactly the lines as in Theorem 2.2.3. This provides a convergence rate similar to Theorem 2.2.3. More specifically, using step size $c/\sqrt{T}$, we get

$$\mathbb{E}[\|f(x_a)\|^2] \leq 2\sqrt{\frac{2(f(x^0) - f(x^*))L}{T}}\sigma. \tag{2.17}$$

The only thing that remains to be proved is that with the step size choice of $\max\{c/\sqrt{T}, \mu_1/(Ln^{2/3})\}$, the minimum of two bounds hold. Consider the case $c/\sqrt{T} > \mu_1/(Ln^{2/3})$. In this case, we have the following:

$$\begin{aligned}
\frac{2\sqrt{\frac{2(f(x^0)-f(x^*))L}{T}}\sigma}{\frac{Ln^{2/3}[f(x^0)-f(x^*)]}{T\nu_1}} &= \frac{2\nu_1\sigma\sqrt{2LT}}{Ln^{2/3}\sqrt{f(x^0) - f(x^*)}} \\
&\leq 2\nu_1/\mu_1 \leq \bar{\nu} := \max\left\{\frac{2\nu_1}{\mu_1}, \frac{\mu_1}{2\nu_1}\right\},
\end{aligned}$$

where ν_1 is the constant in Corollary 2.3.2.2. This inequality holds since $c/\sqrt{T} > \mu_1/(Ln^{2/3})$. Rearranging the above inequality, we have

$$2\sqrt{\frac{2(f(x^0) - f(x^*))L}{T}}\sigma \leq \frac{\bar{\nu}Ln^{2/3}[f(x^0) - f(x^*)]}{T}$$

in this case. Note that the left hand side of the above inequality is precisely the bound obtained by using step size $c/\sqrt{T}$ (see Equation (2.17)). Similarly, the inequality holds in the other direction when $c/\sqrt{T} \leq \mu_1/(Ln^{2/3})$. Using these two observations, we have the desired result. □

2.16 Key Lemmatta

Lemma 2.16.1. *For the intermediate iterates v_t^{s+1} computed by Algorithm 2, we have the following:*

$$\mathbb{E}[\|v_t^{s+1}\|^2] \leq 2\mathbb{E}[\|\nabla f(x_t^{s+1})\|^2] + 2L^2\mathbb{E}[\|x_t^{s+1} - \tilde{x}^s\|^2].$$

Proof. The proof simply follows from the proof of Lemma 2.16.2 with $I_t = \{i_t\}$. □

We now present a result to bound the variance of mini-batch SVRG.

Lemma 2.16.2. *Let u_t^{s+1} be computed by the mini-batch version of Algorithm 2 i.e., Algorithm 4 with mini-batch size b. Then,*

$$\mathbb{E}[\|u_t^{s+1}\|^2] \leq 2\mathbb{E}[\|\nabla f(x_t^{s+1})\|^2] + \tfrac{2L^2}{b}\mathbb{E}[\|x_t^{s+1} - \tilde{x}^s\|^2].$$

Proof. For the ease of exposition, we use the following notation:

$$\zeta_t^{s+1} = \frac{1}{|I_t|}\sum_{i_t \in I_t}\left(\nabla f_{i_t}(x_t^{s+1}) - \nabla f_{i_t}(\tilde{x}^s)\right).$$

We use the definition of u_t^{s+1} to get

$$\begin{aligned}
&\mathbb{E}[\|u_t^{s+1}\|^2] = \mathbb{E}[\|\zeta_t^{s+1} + \nabla f(\tilde{x}^s)\|^2] \\
&= \mathbb{E}[\|\zeta_t^{s+1} + \nabla f(\tilde{x}^s) - \nabla f(x_t^{s+1}) + \nabla f(x_t^{s+1})\|^2] \\
&\leq 2\mathbb{E}[\|\nabla f(x_t^{s+1})\|^2] + 2\mathbb{E}[\|\zeta_t^{s+1} - \mathbb{E}[\zeta_t^{s+1}]\|^2] \\
&= 2\mathbb{E}[\|\nabla f(x_t^{s+1})\|^2] + \frac{2}{b^2}\mathbb{E}\left[\left\|\sum_{i_t \in I_t}\left(\nabla f_{i_t}(x_t^{s+1}) - \nabla f_{i_t}(\tilde{x}^s) - \mathbb{E}[\zeta_t^{s+1}]\right)\right\|^2\right]
\end{aligned}$$

The first inequality follows from Lemma 2.18.4 (with $r = 2$) and the fact that $\mathbb{E}[\zeta_t^{s+1}] = \nabla f(x_t^{s+1}) - \nabla f(\tilde{x}^s)$. From the above inequality, we get

$$\begin{aligned}
&\mathbb{E}[\|u_t^{s+1}\|^2] \\
&\leq 2\mathbb{E}[\|\nabla f(x_t^{s+1})\|^2] + \frac{2}{b^2}\mathbb{E}\left[\sum_{i_t \in I_t}\|\nabla f_{i_t}(x_t^{s+1}) - \nabla f_{i_t}(\tilde{x}^s)\|^2\right] \\
&\leq 2\mathbb{E}[\|\nabla f(x_t^{s+1})\|^2] + \frac{2L^2}{b}\mathbb{E}[\|x_t^{s+1} - \tilde{x}^s\|^2]
\end{aligned}$$

The first inequality follows from Lemma 2.18.3 and noting that for a random variable ζ, $\mathbb{E}[\|\zeta - \mathbb{E}[\zeta]\|^2] \leq \mathbb{E}[\|\zeta\|^2]$. The last inequality follows from L-smoothness of f_{i_t}. □

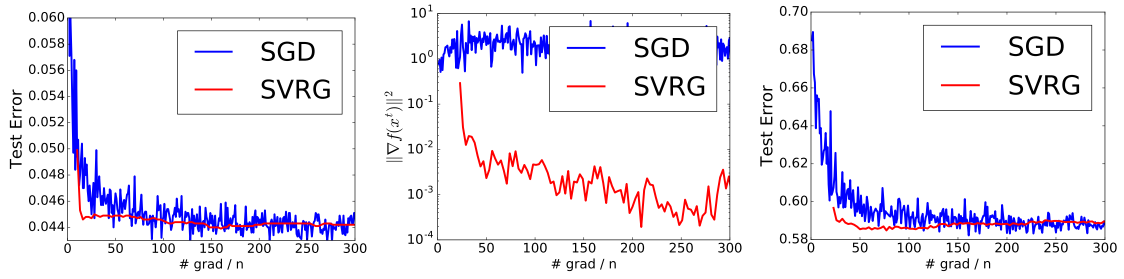


Figure 2.2: Neural network results for MNIST and STL-10. The leftmost result is for MNIST. The remaining two plots are of STL-10.

2.17 Experiments

Figure 2.2 shows the remaining plots for MNIST and STL-10 datasets. As seen in the plots, there is no significant difference in the test error of SVRG and SGD for these datasets.

2.18 Other Lemmas

We need Lemma 2.18.1 for our results in the convex case.

Lemma 2.18.1 (Johnson and Zhang [71]). *Let $g : \mathbb{R}^d \to \mathbb{R}$ be convex with L-Lipschitz continuous gradient. Then,*

$$\|\nabla g(x) - \nabla g(y)\|^2 \leq 2L[g(x) - g(y) - \langle \nabla g(y), x - y \rangle],$$

for all $x, y \in \mathbb{R}^d$.

Proof. Consider $h(x) := g(x) - g(y) - \langle \nabla g(y), x - y \rangle$ for arbitrary $y \in \mathbb{R}^d$. Observe that ∇h is also L-Lipschitz continuous. Note that $h(x) \geq 0$ (since $h(y) = 0$ and $\nabla h(y) = 0$, or alternatively since h defines a Bregman divergence), from which it follows that

$$\begin{aligned} 0 &\leq \min_\rho [h(x - \rho \nabla h(x))] \\ &\leq \min_\rho [h(x) - \rho \|\nabla h(x)\|^2 + \tfrac{L\rho^2}{2} \|\nabla h(x)\|^2] \\ &= h(x) - \tfrac{1}{2L} \|\nabla h(x)\|^2. \end{aligned}$$

Rewriting in terms of g we obtain the required result. □

Lemma 2.18.2 bounds the variance of SVRG for the convex case. Please refer to [71] for more details.

Lemma 2.18.2 ([71]). *Suppose f_i is convex for all $i \in [n]$. For the updates in Algorithm 2 we have the following inequality:*

$$\mathbb{E}[\|v_t^{s+1}\|^2] \leq 4L[f(x_t^{s+1}) - f(x^*) + f(\tilde{x}^s) - f(x^*)].$$

Proof. The proof follows upon observing the following:

$$\begin{aligned}
&\mathbb{E}[\|v_t^{s+1}\|^2 = \mathbb{E}[\|\nabla f_{i_t}(x_t^{s+1}) - \nabla f_{i_t}(x_0^{s+1}) + \nabla f(\tilde{x}^s)\|^2] \\
&\leq 2\mathbb{E}[\|\nabla f_{i_t}(x_t^{s+1}) - \nabla f_{i_t}(x^*)\|^2] + 2\mathbb{E}[\|\nabla f_{i_t}(\tilde{x}^s) - \nabla f_{i_t}(x^*) - (\nabla f(\tilde{x}^s) - \nabla f(x^*))\|^2] \\
&\leq 2\mathbb{E}[\|\nabla f_{i_t}(x_t^{s+1}) - \nabla f_{i_t}(x^*)\|^2] + 2\mathbb{E}[\|\nabla f_{i_t}(\tilde{x}^s) - \nabla f_{i_t}(x^*)\|^2] \\
&\leq 4L[f(x_t^{s+1} - f(x^*) + f(\tilde{x}^s) - f(x^*)].
\end{aligned}$$

The first inequality follows from Cauchy-Schwarz and Young inequality; the second one from $\mathbb{E}[\|\xi - \mathbb{E}[\xi]\|^2] \leq \mathbb{E}[\|\xi\|^2]$, and the third one from Lemma 2.18.1. □

Lemma 2.18.3. *For random variables $z_1, \ldots, z_r$ are independent and mean 0, we have*

$$\mathbb{E}\left[\|z_1 + ... + z_r\|^2\right] = \mathbb{E}\left[\|z_1\|^2 + ... + \|z_r\|^2\right].$$

Proof. We have the following:

$$\mathbb{E}\left[\|z_1 + ... + z_r\|^2\right] = \sum_{i,i=1}^{r} \mathbb{E}\left[z_i z_j\right] = \mathbb{E}\left[\|z_1\|^2 + ... + \|z_r\|^2\right].$$

The second equality follows from the fact that z_i's are independent and mean 0. □

Lemma 2.18.4. *For random variables $z_1, \ldots, z_r$, we have*

$$\mathbb{E}\left[\|z_1 + ... + z_r\|^2\right] \leq r\mathbb{E}\left[\|z_1\|^2 + ... + \|z_r\|^2\right].$$

Chapter 3

Fast Incremental Methods for Smooth Nonconvex Optimization

3.1 Introduction

In this chapter, we continue our study of the *finite-sum* optimization problem

$$\min_{x \in \mathbb{R}^d} f(x) := \frac{1}{n} \sum_{i=1}^{n} f_i(x), \tag{3.1}$$

where each f_i ($i \in \{1, \dots, n\} \triangleq [n]$) can be nonconvex. Recall that we denote the class of such instances of (3.1) by $\mathcal{F}_n$. While the previous chapter presents a fast variance-reduced algorithm, SVRG, for this problem setting, it is not fully *incremental* since it requires computing the *full gradient* periodically. Our focus in this chapter is to develop incremental methods for smooth nonconvex optimization.

As mentioned earlier, problems of the form (3.1) are of central importance in machine learning where they occur as instances of empirical risk minimization; well-known examples include logistic regression (convex) [68] and deep neural networks (nonconvex) [36]. Consequently, such problems have been intensively studied. A basic approach for solving (3.1) is gradient descent (GD), described by the following:

$$x^{t+1} = x^t - \eta_t \nabla f(x^t), \text{ where } \eta_t > 0. \tag{3.2}$$

However, iteration (10.8) is prohibitively expensive in large-scale settings where n is very large. For such settings, stochastic and incremental methods are typical [18, 33]. These methods use cheap *noisy* estimates of the gradient at each iteration of (10.8) instead of $\nabla f(x^t)$. A particularly popular approach, stochastic gradient descent (SGD) uses ∇f_{i_t}, where i_t in chosen uniformly randomly from $\{1, \dots, n\}$. While the cost of each iteration is now greatly reduced, it is not without any drawbacks. Due to the *noise* (also known as variance in stochastic methods) in the gradients, one has to typically use decreasing step-sizes η_t to ensure convergence, and consequently, the convergence rate gets adversely affected.

Therefore, it is of great interest to overcome the slowdown of SGD without giving up its scalability. Towards this end, for convex instances of (3.1), remarkable progress has been made recently. The key realization is that if we make multiple passes through the data, we can store information that allows us to reduce variance of the stochastic gradients. As a result, we can use constant stepsizes (rather than diminishing scalars) and obtain convergence faster than SGD, both in theory and practice [33, 71, 153]. Nonconvex instances of (3.1) are also known to enjoy similar speedups [71], but existing analysis does not explain this success as it relies heavily on convexity to control variance. Since SGD dominates large-scale nonconvex optimization (including neural network training), it is of great value to develop faster nonconvex stochastic methods.

In this chapter, we analyze a variant of SAGA [33] algorithm, an incremental aggregated gradient algorithm that extends the seminal SAG method of [153], and has been shown to work well for convex finite-sum problems [33]. Specifically, we analyze SAGA for the class $\mathcal{F}_n$ using the *incremental first-order oracle* (IFO) [2]. Recall that for $f \in \mathcal{F}_n$, an IFO is a subroutine that takes an index $i \in [n]$ and a point $x \in \mathbb{R}^d$, and returns the pair $(f_i(x), \nabla f_i(x))$. To our knowledge, this work presents the first analysis of fast convergence for an incremental aggregated gradient method for *nonconvex* problems. The attained rates improve over both SGD and GD, a benefit that is also corroborated by experiments.

3.1.1 Related work

A concise survey of incremental gradient methods is [18]. An accessible analysis of stochastic convex optimization ($\min \mathbb{E}_z[F(x,z)]$) is [110]. Classically, SGD stems from the seminal work [148], and has since witnessed many developments [77], including parallel and distributed variants [3, 15, 128], though non-asymptotic convergence analysis is limited to convex setups. Faster rates for convex problems in $\mathcal{F}_n$ are attained by variance reduced methods, e.g., [33, 71, 133, 153, 156]. Linear convergence of stochastic dual coordinate ascent when f_i ($i \in [n]$) may be nonconvex but f is strongly convex is studied in [154]. Lower bounds for convex finite-sum problems are studied in [2].

For nonconvex nonsmooth problems the first incremental proximal-splitting methods is in [164], though only asymptotic convergence is studied. Hong [60] studies convergence of a distributed nonconvex incremental ADMM algorithm. The first work to present non-asymptotic convergence rates for SGD is [47]; this work presents an $O(1/\epsilon^2)$ iteration bound for SGD to satisfy approximate stationarity $\|\nabla f(x)\|^2 \leq \epsilon$, and their convergence criterion is motivated by the gradient descent analysis of Nesterov [111]. The first analysis for nonconvex variance reduced stochastic gradient is due to [160], who apply it to the specific problem of principal component analysis (PCA).

3.2 Preliminaries

In this section, we further explain our assumptions and goals. Recall that a function f is *L-smooth* if there is a constant L such that $\|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\|, \quad \forall\, x, y \in \mathbb{R}^d$.

Throughout, we assume that all f_i in (11.1) are L-smooth, i.e., $\|\nabla f_i(x) - \nabla f_i(y)\| \leq L\|x - y\|$ for all $i \in [n]$. Such an assumption is common in the analysis of first-order methods. For ease of exposition, we assume the Lipschitz constant L to be independent of n. For our analysis, we also discuss the class of τ-*gradient dominated* [115, 122] functions, namely functions for which

$$f(x) - f(x^*) \leq \tau\|\nabla f(x)\|^2, \tag{3.3}$$

where x^* is a global minimizer of f. This class of functions was originally introduced by Polyak in [122]. Observe that such functions need not be convex. Also notice that gradient dominance (3.3) is a restriction on the overall function f, but not on the individual functions f_i ($i \in [n]$).

Following [47, 111] we use $\|\nabla f(x)\|^2 \leq \epsilon$ to judge approximate stationarity of x. Contrast this with SGD for convex f, where one uses $[f(x) - f(x^*)]$ or $\|x - x^*\|^2$ as criteria for convergence analysis. Such criteria cannot be used for nonconvex functions due to the intractability of the problem.

While the quantities $\|\nabla f(x)\|^2$, $[f(x) - f(x^*)]$, or $\|x - x^*\|^2$ are not comparable in general, they are typically assumed to be of similar magnitude (see [47]). We note that our analysis does *not* assume n to be a constant, so we report dependence on it in our results. Furthermore, while stating our complexity results, we assume that the initial point of the algorithms is constant, i.e., $f(x^0) - f(x^*)$ and $\|x^0 - x^*\|$ are constants. For our analysis, we will need the following definition.

Definition 3.2.1. *A point x is called ϵ-accurate if $\|\nabla f(x)\|^2 \leq \epsilon$. A stochastic iterative algorithm is said to achieve ϵ-accuracy in t iterations if $\mathbb{E}[\|\nabla f(x^t)\|^2] \leq \epsilon$, where the expectation is taken over its stochasticity.*

We start our discussion of algorithms by recalling SGD, which performs the following update in its t^{th} iteration:

$$x^{t+1} = x^t - \eta_t \nabla f_{i_t}(x), \tag{3.4}$$

where i_t is a random index chosen from $[n]$, and the gradient $\nabla f_{i_t}(x^t)$ approximates the gradient of f at x^t, $\nabla f(x^t)$. It can be seen that the update is unbiased, i.e., $\mathbb{E}[\nabla f_{i_t}(x^t)] = \nabla f(x^t)$ since the index i_t is chosen uniformly at random. Though the SGD update is unbiased, it suffers from variance due to the aforementioned stochasticity in the chosen index. To control the variance one has to decrease the step size η_t in (3.4), which in turn leads to slow convergence. The following is a well-known result on SGD in the context of nonconvex optimization [47].

Theorem 3.2.2. *Suppose $\|\nabla f_i\| \leq \sigma$ i.e., gradient of function f_i is bounded for all $i \in [n]$, then the IFO complexity of* SGD *to obtain an ϵ-accurate solution is $O(1/\epsilon^2)$.*

It is instructive to compare the above result with the convergence rate of GD. The IFO complexity of GD is $O(n/\epsilon)$. Thus, while SGD eliminates the dependence on n, the convergence rate worsens to $O(1/\epsilon^2)$ from $O(1/\epsilon)$ in GD. In the next section, we investigate an incremental method with faster convergence.

Algorithm 5: SAGA(x^0, T, η)

```
1: Input: $x^0 \in \mathbb{R}^d$, $\alpha_i^0 = x^0$ for $i \in [n]$, number of iterations $T$, step size $\eta > 0$
2: $g^0 = \frac{1}{n}\sum_{i=1}^n \nabla f_i(\alpha_i^0)$
3: for $t = 0$ to $T-1$ do
4:     Uniformly randomly pick $i_t, j_t$ from $[n]$
5:     $v^t = \nabla f_{i_t}(x^t) - \nabla f_{i_t}(\alpha_{i_t}^t) + g^t$
6:     $x^{t+1} = x^t - \eta v^t$
7:     $\alpha_{j_t}^{t+1} = x^t$ and $\alpha_j^{t+1} = \alpha_j^t$ for $j \neq j_t$
8:     $g^{t+1} = g^t - \frac{1}{n}(\nabla f_{j_t}(\alpha_{j_t}^t) - \nabla f_{j_t}(\alpha_{j_t}^{t+1}))$
9: end for
10: Output: Iterate $x_a$ chosen uniformly random from $\{x^t\}_{t=0}^{T-1}$.
```

3.3 Algorithm

We describe below the SAGA algorithm and prove its fast convergence for *nonconvex* optimization. SAGA is a popular incremental method in machine learning and optimization communities. It is very effective in reducing the variance introduced due to stochasticity in SGD. Algorithm 5 presents pseudocode for SAGA. Note that the update v^t (Line 5) is unbiased, i.e., $\mathbb{E}[v^t] = \nabla f(x^t)$. This is due to the uniform random selection of index i_t. It can be seen in Algorithm 5 that SAGA maintains gradients at α_i for $i \in [n]$. This additional set is critical to reducing the variance of the update v^t. At each iteration of the algorithm, one of the α_i is updated to the current iterate. An implementation of SAGA requires storage and updating of gradients $\nabla f_i(\alpha_i)$; the storage cost of the algorithm is nd. While this leads to higher storage in comparison to SGD, this cost is often reasonable for many applications. Furthermore, this cost can be reduced in the case of specific models; refer to [33] for more details.

For ease of exposition, we introduce the following quantity:

$$\Gamma_t = \left(\eta - \frac{c_{t+1}\eta}{\beta} - \eta^2 L - 2c_{t+1}\eta^2\right), \tag{3.5}$$

where the parameters c_{t+1}, β and η will be defined shortly. We start with the following set of key results that provide convergence rate of Algorithm 5.

Lemma 3.3.1. *For $c_t, c_{t+1}, \beta > 0$, suppose we have*

$$c_t = c_{t+1}(1 - \tfrac{1}{n} + \eta\beta + 2\eta^2 L^2) + \eta^2 L^3.$$

Also let η, β and c_{t+1} be chosen such that $\Gamma_t > 0$. Then, the iterates $\{x^t\}$ of Algorithm 5 satisfy the bound

$$\mathbb{E}[\|\nabla f(x^t)\|^2] \leq \frac{R^t - R^{t+1}}{\Gamma_t},$$

where $R^t := \mathbb{E}[f(x^t) + (c_t/n)\sum_{i=1}^n \|x^t - \alpha_i^t\|^2]$.

The proof of this lemma is given in Section 3.9. Using this lemma we prove the following result on the iterates of SAGA.

Theorem 3.3.2. *Let $f \in \mathcal{F}_n$. Let $c_T = 0$, $\beta > 0$, and $c_t = c_{t+1}(1 - \frac{1}{n} + \eta\beta + 2\eta^2 L^2) + \eta^2 L^3$ be such that $\Gamma_t > 0$ for $0 \leq t \leq T-1$. Define the quantity $\gamma_n := \min_{0 \leq t \leq T-1} \Gamma_t$. Then the output x_a of Algorithm 5 satisfies the bound*

$$\mathbb{E}[\|\nabla f(x_a)\|^2] \leq \frac{f(x^0) - f(x^*)}{T\gamma_n},$$

where x^* is an optimal solution to (3.1)*.*

Proof. We apply telescoping sums to the result of Lemma 3.3.1 to obtain

$$\begin{aligned}\gamma_n \textstyle\sum_{t=0}^{T-1} \mathbb{E}[\|\nabla f(x^t)\|^2] &\leq \textstyle\sum_{t=0}^{T-1} \Gamma_t \mathbb{E}[\|\nabla f(x^t)\|^2] \\ &\leq R^0 - R^T.\end{aligned}$$

The first inequality follows from the definition of γ_n. This inequality in turn implies the bound

$$\textstyle\sum_{t=0}^{T-1} \mathbb{E}[\|\nabla f(x^t)\|^2] \leq \frac{\mathbb{E}[f(x^0) - f(x^T)]}{\gamma_n}, \tag{3.6}$$

where we used that $R^T = \mathbb{E}[f(x^T)]$ (since $c_T = 0$), and that $R^0 = \mathbb{E}[f(x^0)]$ (since $\alpha_i^0 = x^0$ for $i \in [n]$). Using inequality (3.6), the optimality of x^*, and the definition of x_a in Algorithm 5, we obtain the desired result. □

Note that the notation γ_n involves n, since this quantity can depend on n. To obtain an explicit dependence on n, we have to use an appropriate choice of β and η. This is made precise by the following main result of the chapter.

Theorem 3.3.3. *Suppose $f \in \mathcal{F}_n$. Let $\eta = 1/(3Ln^{2/3})$ and $\beta = L/n^{1/3}$. Then, $\gamma_n \geq \frac{1}{12Ln^{2/3}}$ and we have the bound*

$$\mathbb{E}[\|\nabla f(x_a)\|^2] \leq \frac{12Ln^{2/3}[f(x^0) - f(x^*)]}{T},$$

where x^* is an optimal solution to the problem in (3.1) *and x_a is the output of Algorithm 5.*

Proof. With the values of η and β, let us first establish an upper bound on c_t. Let θ denote $\frac{1}{n} - \eta\beta - 2\eta^2 L^2$. Observe that $\theta < 1$ and $\theta \geq 4/(9n)$. This is due to the specific values of η and β stated in the theorem. Also, we have $c_t = c_{t+1}(1-\theta) + \eta^2 L^3$. Using this relationship, it is easy to see that $c_t = \eta^2 L^3 \frac{1-(1-\theta)^{T-t}}{\theta}$. Therefore, we obtain the bound

$$c_t = \eta^2 L^3 \tfrac{1-(1-\theta)^{T-t}}{\theta} \leq \frac{\eta^2 L^3}{\theta} \leq \frac{L}{4n^{1/3}}, \tag{3.7}$$

for all $0 \leq t \leq T$, where the inequality follows from the definition of η and the fact that $\theta \geq 4/(9n)$. Using the above upper bound on c_t we can conclude that

$$\gamma_n = \min_t \big(\eta - \tfrac{c_{t+1}\eta}{\beta} - \eta^2 L - 2c_{t+1}\eta^2\big) \geq \frac{1}{12Ln^{2/3}},$$

upon using the following inequalities: (i) $c_{t+1}\eta/\beta \leq \eta/4$, (ii) $\eta^2 L \leq \eta/3$ and (iii) $2c_{t+1}\eta^2 \leq \eta/6$, which hold due to the upper bound on c_t in (3.7). Substituting this bound on γ_n in Theorem 3.3.2, we obtain the desired result. □

A more general result with step size $\eta < 1/(3Ln^{2/3})$ can be proved, but it will only lead to a theoretically suboptimal convergence result. Recall that each iteration of Algorithm 5 requires $O(1)$ IFO calls. Using this fact, we can rewrite Theorem 3.3.3 in terms of its IFO complexity as follows.

Corollary 3.3.3.1. *If $f \in \mathcal{F}_n$, then the IFO complexity of Algorithm 5 (with parameters from Theorem 3.3.3) to obtain an ϵ-accurate solution is $O(n + n^{2/3}/\epsilon)$.*

This corollary follows from the $O(1)$ per iteration cost of Algorithm 5 and because n IFO calls are required to calculate g^0 at the start of the algorithm. In special cases, the initial $O(n)$ IFO calls can be avoided (refer to [33, 153] for details). By comparing the IFO complexity of SAGA ($O(n + n^{2/3}/\epsilon)$) with that of GD ($O(n/\epsilon)$), we see that SAGA is faster than GD by a factor of $n^{1/3}$.

3.4 Finite Sums with Regularization

In this section, we study the problem of finite-sum problems with additional regularization. More specifically, we consider problems of the form

$$\min_{x \in \mathbb{R}^d} f(x) := \frac{1}{n}\sum_{i=1}^{n} f_i(x) + r(x), \tag{3.8}$$

where $r : \mathbb{R}^d \to \mathbb{R}$ is an L-smooth (possibly nonconvex) function. Problems of this nature arise in machine learning where the functions f_i are loss functions and r is a regularizer. Since we assumed r to be smooth, (3.8) can be reformulated as (3.1) by simply encapsulating r into the functions f_i. However, as we will see, it is beneficial to handle the regularization separately. We call the variant of SAGA with the additional regularization as REG-SAGA. The key difference between REG-SAGA and SAGA lies in Line 6 of Algorithm 5. In particular, for REG-SAGA, Line 6 of Algorithm 5 is replaced with the following update:

$$x^{t+1} = x^t - \eta(v^t + \nabla r(x^t)). \tag{3.9}$$

Note that the primary difference is that the part of gradient based on function r is updated at each iteration of the algorithm. The convergence of REG-SAGA can be proved along the lines of our analysis of SAGA. Hence, we omit the details for brevity and directly state the following key result stating the IFO complexity of REG-SAGA.

Theorem 3.4.1. *If function f is of the form in* (3.8)*, then the IFO complexity of* REG-SAGA *to obtain an ϵ-accurate solution is $O(n + n^{2/3}/\epsilon)$.*

The proof essentially follows along the lines of the proof of Theorem 3.3.3. The difference, however, being that the update corresponding to function $r(x)$ is handled explicitly at each iteration. Note that the above IFO complexity is not different from that in Corollary 3.3.3.1. However, its main benefit comes in the form of storage efficiency in problems with more structure. To understand this, consider the problems of form

$$\min_{x \in \mathbb{R}^d} f(x) := \frac{1}{n} \sum_{i=1}^{n} l(x^\top z_i) + r(x), \tag{3.10}$$

where $z_i \in \mathbb{R}^d$ for $i \in [n]$ while $l : \mathbb{R} \to \mathbb{R}_{\geq 0}$ is a differentiable loss function. Here, l and r can be in general nonconvex. Such problems are popularly known as (regularized) empirical risk minimization in machine learning literature. We can directly apply SAGA to (3.10) by casting it in the form (3.1). However, recall that the storage cost of SAGA is $O(nd)$ due to the cost of storing the gradient at α_i^t. This storage cost can be avoided in REG-SAGA by handling the function r separately. Indeed, for REG-SAGA we need to store just $\nabla l(x^\top z_i)$ for all $i \in [n]$ (as $\nabla r(x)$ is updated at each iteration). By observing that $\nabla l(x^\top z_i) = l'(x^\top z_i) z_i$, where l' represents the derivative of l, it is apparent that we need to store only the scalars $l'(x^\top z_i)$ for REG-SAGA. This reduces the storage $O(n)$ instead of $O(nd)$ in SAGA.

3.5 Gradient Dominated Functions

Until now the only assumption we used was Lipschitz continuity of gradients. An immediate question is whether the IFO complexity can be further improved under stronger assumptions. We provide an affirmative answer to this question by showing that for gradient dominated functions, a variant of SAGA attains linear convergence rate. Recall that a function f is called τ-gradient dominated if around an optimal point x^*, f satisfies the following growth condition:

$$f(x) - f(x^*) \leq \tau \|\nabla f(x)\|^2, \quad \forall x \in \mathbb{R}^d.$$

For such functions, we use the variant of SAGA shown in Algorithm 6. Observe that Algorithm 6 uses SAGA as a subroutine. Alternatively, one can rewrite Algorithm 6 in the form of KT iterations of Algorithm 5 where one updates $\{\alpha_i\}$ after every T iterations and then selects a random iterate amongst the last T iterates. For this variant of SAGA, we can prove the following linear convergence result.

Theorem 3.5.1. *If f is τ-gradient dominated, then with $\eta = 1/(3Ln^{2/3})$ and $T = \lceil 24L\tau n^{2/3} \rceil$, the iterates of Algorithm 6 satisfy $\mathbb{E}[\|f(x^k)\|^2] \leq 2^{-k}\|f(x^0)\|^2$, where x^* is an optimal solution of* (3.1).

Algorithm 6: GD-SAGA(x^0, K, T, η)

```
1: Input: $x^0 \in \mathbb{R}^d$, $K$, epoch length $m$, step sizes $\eta > 0$
2: for $k = 0$ to $K$ do
3:     $x^k = \text{SAGA}(x^{k-1}, T, \eta)$
4: end for
5: Output: $x^K$
```

Proof. The iterates of Algorithm 6 satisfy the bound

$$\mathbb{E}[\|\nabla f(x^k)\|^2] \leq \frac{\mathbb{E}[f(x^{k-1}) - f(x^*)]}{2\tau}, \tag{3.11}$$

which holds due to Theorem 3.3.3 given the choices of η and T assumed in the statement. However, f is τ-gradient dominated, so $\mathbb{E}[\|\nabla f(x^{k-1})\|^2] \geq \mathbb{E}[f(x^{k-1}) - f(x^*)]/\tau$, which combined with (3.11) concludes the proof. □

An immediate consequence of this theorem is the following.

Corollary 3.5.1.1. *If f is τ-gradient dominated, the IFO complexity of Algorithm 6 (with parameters from Theorem 3.5.1) to compute an ϵ-accurate solution is $O((n + \tau n^{2/3}) \log(1/\epsilon))$.*

While we state the result in terms of $\|\nabla f(x)\|^2$, it is not hard to see that for gradient dominated functions a similar result holds for the convergence criterion being $[f(x) - f(x^*)]$.

Theorem 3.5.2. *If f is τ-gradient dominated, with $\eta = 1/(3Ln^{2/3})$ and $T = \lceil 24L\tau n^{2/3} \rceil$, the iterates $\{x^k\}$ of Algorithm 6 satisfy*

$$\mathbb{E}[f(x^k) - f(x^*)] \leq 2^{-k}[f(x^0) - f(x^*)],$$

where x^* is an optimal solution to (3.1).

A noteworthy aspect of the above result is the linear convergence rate to a *global* optimum. Therefore, the above result is stronger than Theorem 3.3.3. Note that throughout our analysis of gradient dominated functions, no assumptions other than Lipschitz smoothness are placed on the *individual* set of functions f_i. We emphasize here that these results can be further improved with additional assumptions (e.g., strong convexity) on the individual functions f_i and on f. Also note that GD can achieve linear convergence rate for gradient dominated functions [122]. However, the IFO complexity of GD is $O(\tau n \log(1/\epsilon))$, which is strictly worse than IFO complexity of GD-SAGA (see Corollary 3.5.1.1).

3.6 Minibatch Variant

A common variant of incremental methods is to sample a set of indices I_t instead of single index i_t when approximating the gradient. Such a variant is generally referred to as

Algorithm 7: Minibatch-SAGA(x^0, b, T, η)

```
1: Input: $x^0 \in \mathbb{R}^d$, $\alpha_i^0 = x^0$ for $i \in [n]$, minibatch size $b$, number of iterations $T$, step size $\eta > 0$
2: $g^0 = \frac{1}{n}\sum_{i=1}^n \nabla f_i(\alpha_i^0)$
3: for $t = 0$ to $T-1$ do
4:     Uniformly randomly pick (with replacement) indices sets $I_t$, $J_t$ of size $b$ from $[n]$
5:     $v^t = \frac{1}{|I_t|}\sum_{i\in I_t}(\nabla f_i(x^t) - \nabla f_i(\alpha_{i_t}^t)) + g^t$
6:     $x^{t+1} = x^t - \eta v^t$
7:     $\alpha_j^{t+1} = x^t$ for $j \in J_t$ and $\alpha_j^{t+1} = \alpha_j^t$ for $j \notin J_t$
8:     $g^{t+1} = g^t - \frac{1}{n}\sum_{j\in J_t}(\nabla f_j(\alpha_j^t) - \nabla f_j(\alpha_j^{t+1}))$
9: end for
10: Output: Iterate $x_a$ chosen uniformly random from $\{x^t\}_{t=0}^{T-1}$.
```

a "minibatch" version of the algorithm. Minibatch variants are of great practical significance since they reduce the variance of incremental methods and promote parallelism. Algorithm 7 lists the pseudocode for a minibatch variant of SAGA. Algorithm 7 uses a set I_t of size $|I_t| = b$ for calculating the update v^t instead of a single index i_t used in Algorithm 5. By using a larger b, one can reduce the variance due to the stochasticity in the algorithm. Such a procedure is also beneficial in parallel settings since the calculation of the update v^t can be parallelized. For this algorithm, we can prove the following convergence result.

Theorem 3.6.1. *Suppose $f \in \mathcal{F}_n$. Let $\eta = b/(3Ln^{2/3})$ and $\beta = L/n^{1/3}$. Then for the output x_a of Algorithm 7 (with $b < n^{2/3}$) we have $\gamma_n \geq \frac{b}{12Ln^{2/3}}$ and*

$$\mathbb{E}[\|\nabla f(x_a)\|^2] \leq \frac{12Ln^{2/3}[f(x^0) - f(x^*)]}{bT},$$

where x^* is an optimal solution to (3.1).

We omit the details of the proof since it is similar to the proof of Theorem 3.3.3. Note that the key difference in comparison to Theorem 5 is that we can now use a more aggressive step size $\eta = b/(3Ln^{2/3})$ due to a larger minibatch size b. An interesting aspect of the result is the $O(1/b)$ dependence on the minibatch size b. As long as this size is not large ($b < n^{2/3}$), one can significantly improve the convergence rate to a stationary point. A restatement of aforementioned result in terms of IFO complexity is provided below.

Corollary 3.6.1.1. *If $f \in \mathcal{F}_n$, then the IFO complexity of Algorithm 7 (with parameters from Theorem 3.6.1 and minibatch size $b < n^{2/3}$) to obtain an ϵ-accurate solution is $O(n + n^{2/3}/\epsilon)$.*

By comparing the above result with Corollary 3.3.3.1, we can see that the IFO complexity of minibatch-SAGA is the same SAGA. However, since the b gradients can be computed in parallel, one can achieve (theoretical) b times speedup in multicore and distributed settings. In contrast, the performance SGD degrades with minibatch size b since the improvement in convergence rate for SGD is typically $O(1/\sqrt{b})$ but b IFO calls are required at each iteration of minibatch-SGD. Thus, SAGA has a much more efficient minibatch version in comparison to SGD.

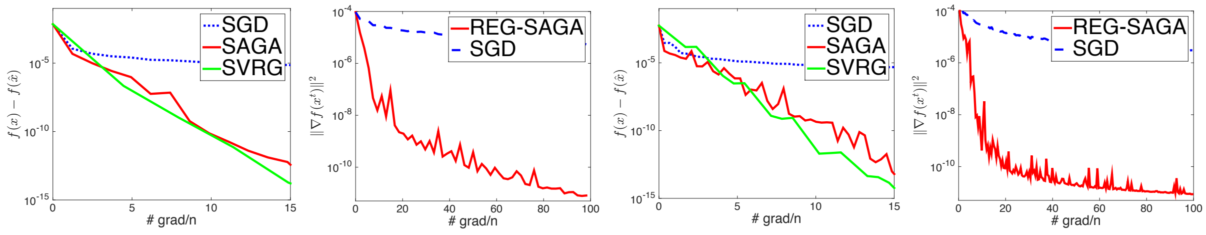


Figure 3.1: Results for nonconvex regularized generalized linear models (see Equation (3.12)). The first and last two figures correspond to rcv1 and realsim datasets respectively. The results compare the performance of REG-SAGA and SGD algorithms. Here $\hat{x}$ corresponds to the solution obtained by running GD for a very long time and using multiple restarts. As seen in the figure, REG-SAGA converges much faster than SGD in terms of objective function value and the stationarity gap $\|\nabla f(x)\|^2$.

3.7 Experiments

We present our empirical results in this section. For our experiments, we study the problem of binary classification using nonconvex regularizers. The input consists of tuples $\{(z_i, y_i)\}_{i=1}^n$ where $z_i \in \mathbb{R}^d$ (commonly referred to as features) and $y_i \in \{-1, 1\}$ (class labels). We are interested in the empirical loss minimization setup described in Section 3.4. Recall that problem of finite sum with regularization takes the form

$$\min_{x \in \mathbb{R}^d} f(x) := \frac{1}{n} \sum_{i=1}^n f_i(x) + r(x). \tag{3.12}$$

For our experiments, we use logistic function for f_i, i.e., $f_i(x) = \log(1 + \exp(-y_i x^\top z_i))$ for all $i \in [n]$. All z_i are normalized such that $\|z_i\| = 1$. We observe that the loss function has Lipschitz continuous gradients. The function $r(x) = \lambda \sum_{i=1}^d \alpha x_i^2/(1 + \alpha x_i^2)$ is chosen as the regularizer (see [9]). Note that the regularizer is nonconvex and smooth. In our experiments, we use the parameter settings of $\lambda = 0.001$ and $\alpha = 1$ for all the datasets.

We compare the performance of SGD (the *de facto* incremental method for nonconvex optimization) with nonconvex REG-SAGA in our experiments. The comparison is based on the following criteria: (i) the objective function value (also called training loss in this context), which is the main goal of the chapter; and (ii) the stationarity gap $\|\nabla f(x)\|^2$, the criteria used for our theoretical analysis. For the step size of SGD, we use the popular $t-$inverse schedule $\eta_t = \eta_0(1 + \eta' \lfloor t/n \rfloor)^{-1}$, where η_0 and η' are tuned so that SGD gives the best performance on the training loss. In our experiments, we also use $\eta' = 0$; this results in a fixed step size for SGD. For REG-SAGA, a fixed step size is chosen (as suggested by our analysis) so that it gives the best performance on the objective function value, i.e., training loss. Note that due to the structure of the problem in (3.12), as discussed in Section 3.4, the storage cost of REG-SAGA is just $O(n)$.

Initialization is critical to many of the incremental methods like REG-SAGA. This is due to the stronger dependence of the convergence on the initial point (see Theorem 3.3.3). Furthermore, one has to obtain $\nabla f_i(\alpha_i^0)$ for all $i \in [n]$ in REG-SAGA algorithm (see Algorithm 5). For initialization of both SGD and REG-SAGA, we make one pass (without replacement) through the dataset and perform the updates of SGD during this pass. Doing so not only allows us to also obtain a good initial point x^0 but also to compute the initial values of $\nabla f(\alpha_i^0)$ for $i \in [n]$. Note that this choice results in a variant of REG-SAGA where α_i^0 are different for various i (unlike the pseudocode in Algorithm 5). The convergence rates of this variant can be shown to be similar to that of Algorithm 5.

Figure 3.1 shows the results of our experiments. The results are on two standard UCI datasets, 'rcv1' and 'realsim'[1]. The plots compare the criteria mentioned earlier against the number of IFO calls made by the corresponding algorithm. For the objective function, we look at the difference between the objective function ($f(x^t)$) and the best objective function value obtained by running GD for a very large number of iterations using multiple initializations (denoted by $f(\hat{x})$). As shown in the figure, REG-SAGA converges much faster than SGD in terms of objective value. Furthermore, as supported by the theory, the stationarity gap for REG-SAGA is very small in comparison to SGD. Also, in our experiments, the selection of step size was much easier for REG-SAGA when compared to SGD. Overall the empirical results for nonconvex regularized problems are promising. It will be interesting to apply the approach for other smooth nonconvex problems.

3.8 Discussion

Before ending this chapter, it is important to compare and contrast different convergence rates obtained in the chapter. For general smooth nonconvex problems, we observed that SAGA has a low IFO complexity of $O(n + n^{2/3}/\epsilon)$ in comparison to SGD ($O(1/\epsilon^2)$) and GD ($O(n/\epsilon)$). This difference in the convergence is especially significant if one requires a medium to high accuracy solution, i.e., ϵ is small. Furthermore, for gradient dominated functions, where SGD obtains a sublinear convergence rate of $O(1/\epsilon^2)$[2] as opposed to fast linear convergence rate of a variant of SAGA (see Theorem 3.5.1.1). It is an interesting future work to explore other setups where we can achieve stronger convergence rates. Surprisingly, the aforementioned convergence rates of SAGA match with those obtained for SVRG in the previous chapter.

From our analysis of minibatch-SAGA in Section 3.6, we observe that SAGA profits from mini-batching much more than SGD. In particular, one can achieve a (theoretical) linear speedup using mini-batching in SAGA in parallel settings. On the other hand, the performance of SGD typically degrades with minibatching. In summary, SAGA enjoys all the benefits of GD like constant step size, efficient minibatching with much weaker dependence on n.

Notably, SAGA, similar to SVRG and unlike SGD, does not use any additional assumption of bounded gradients (see Theorem 3.2.2 and Corollary 3.3.3.1). Moreover, if

[1]The datasets can be downloaded from https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html.

[2]For SGD, we are not aware of any better convergence rates for gradient dominated functions.

one uses a constant step size for SGD, we need to have an advance knowledge of the total number of iterations T in order to obtain the convergence rate mentioned in Theorem 3.2.2. Although we restrict our attention to SAGA algorithm, by using our proof technique, we can show that another incremental method, SDCA, exhibits very similar properties for the problem setting of our interest.

Appendix: Omitted Proofs

3.9 Proof of Lemma 3.3.1

Proof. Since f is L-smooth, from Lemma 3.9.2, we have

$$\mathbb{E}[f(x^{t+1})] \leq \mathbb{E}[f(x^t) + \langle \nabla f(x^t), x^{t+1} - x^t \rangle + \tfrac{L}{2}\|x^{t+1} - x^t\|^2].$$

We first note that the update in Algorithm 5 is unbiased i.e., $\mathbb{E}[v^t] = \nabla f(x^t)$. By using this property of the update on the right hand side of the inequality above, we get the following:

$$\mathbb{E}[f(x^{t+1})] \leq \mathbb{E}[f(x^t) - \eta_t\|\nabla f(x^t)\|^2 + \tfrac{L\eta^2}{2}\|v^t\|^2]. \tag{3.13}$$

Here we used the fact that $x^{t+1} - x^t = -\eta v^t$ (see Algorithm 5). Consider now the Lyapunov function

$$R^t := \mathbb{E}[f(x^t) + \tfrac{c_t}{n}\sum_{i=1}^{n}\|x^t - \alpha_i^t\|^2].$$

For bounding R^{t+1} we need the following:

$$\frac{1}{n}\sum_{i=1}^{n}\mathbb{E}[\|x^{t+1} - \alpha_i^{t+1}\|^2] = \frac{1}{n}\sum_{i=1}^{n}\left[\frac{1}{n}\mathbb{E}\|x^{t+1} - x^t\|^2 + \frac{n-1}{n}\underbrace{\mathbb{E}\|x^{t+1} - \alpha_i^t\|^2}_{T_1}\right]. \tag{3.14}$$

The above equality from the definition of α_i^{t+1} and the uniform randomness of index j_t in Algorithm 5. The term T_1 in (3.14) can be bounded as follows

$$\begin{aligned}
T_1 &= \mathbb{E}[\|x^{t+1} - x^t + x^t - \alpha_i^t\|^2] \\
&= \mathbb{E}[\|x^{t+1} - x^t\|^2 + \|x^t - \alpha_i^t\|^2 + 2\langle x^{t+1} - x^t, x^t - \alpha_i^t\rangle] \\
&= \mathbb{E}[\|x^{t+1} - x^t\|^2 + \|x^t - \alpha_i^t\|^2] - 2\eta\mathbb{E}[\langle \nabla f(x^t), x^t - \alpha_i^t\rangle] \\
&\leq \mathbb{E}[\|x^{t+1} - x^t\|^2 + \|x^t - \alpha_i^t\|^2] \\
&\qquad + 2\eta\mathbb{E}\left[\tfrac{1}{2\beta}\|\nabla f(x^t)\|^2 + \tfrac{1}{2}\beta\|x^t - \alpha_i^t\|^2\right].
\end{aligned} \tag{3.15}$$

The second equality again follows from the unbiasedness of the update of SAGA. The last inequality follows from a simple application of Cauchy-Schwarz and Young's inequality. Plugging (3.13) and (3.15) into R^{t+1}, we obtain the following bound:

$$\begin{aligned}
R^{t+1} &\leq \mathbb{E}[f(x^t) - \eta\|\nabla f(x^t)\|^2 + \tfrac{L\eta^2}{2}\|v^t\|^2] \\
&+ \mathbb{E}[c_{t+1}\|x^{t+1} - x^t\|^2 + c_{t+1}\frac{n-1}{n^2}\sum_{i=1}^{n}\|x^t - \alpha_i^t\|^2] \\
&+ \frac{2(n-1)c_{t+1}\eta}{n^2}\sum_{i=1}^{n}\mathbb{E}\left[\tfrac{1}{2\beta}\|\nabla f(x^t)\|^2 + \tfrac{1}{2}\beta\|x^t - \alpha_i^t\|^2\right] \\
&\leq \mathbb{E}[f(x^t) - \left(\eta - \tfrac{c_{t+1}\eta}{\beta}\right)\|\nabla f(x^t)\|^2 \\
&\quad + \left(\tfrac{L\eta^2}{2} + c_{t+1}\eta^2\right)\mathbb{E}[\|v^t\|^2] \\
&\quad + \left(\frac{n-1}{n}c_{t+1} + c_{t+1}\eta\beta\right)\frac{1}{n}\sum_{i=1}^{n}\mathbb{E}\left[\|x^t - \alpha_i^t\|^2\right]. \qquad (3.16)
\end{aligned}$$

To further bound the quantity in (3.16), we use Lemma 3.9.1 to bound $\mathbb{E}[\|v^t\|^2]$, so that upon substituting it into (3.16), we obtain

$$\begin{aligned}
R^{t+1} &\leq \mathbb{E}[f(x^t)] \\
&- \left(\eta - \tfrac{c_{t+1}\eta}{\beta} - \eta^2 L - 2c_{t+1}\eta^2\right)\mathbb{E}[\|\nabla f(x^t)\|^2] \\
&+ \left[c_{t+1}(1 - \tfrac{1}{n} + \eta\beta + 2\eta^2L^2) + \eta^2L^3\right]\tfrac{1}{n}\sum_{i=1}^{n}\mathbb{E}\left[\|x^t - \alpha_i^t\|^2\right] \\
&\leq R^t - (\eta - \tfrac{c_{t+1}\eta}{\beta} - \eta^2 L - 2c_{t+1}\eta^2)\mathbb{E}[\|\nabla f(x^t)\|^2].
\end{aligned}$$

The second inequality follows from the definition of c_t i.e., $c_t = c_{t+1}(1 - \frac{1}{n} + \eta\beta + 2\eta^2L^2) + \eta^2L^3$ and R^t specified in the statement, thus concluding the proof. □

Other Lemmas

The following lemma provides a bound on the variance of the update used in SAGA algorithm. More specifically, it bounds the quantity $\mathbb{E}[\|v^t\|^2]$. A more general result for bounding the variance of the minibatch scheme in Algorithm 7 can be proved along similar lines.

Lemma 3.9.1. *Let v^t be computed by Algorithm 5. Then,*

$$\mathbb{E}[\|v^t\|^2] \leq 2\mathbb{E}[\|\nabla f(x^t)\|^2] + \frac{2L^2}{n}\sum_{i=1}^{n}\mathbb{E}[\|x^t - \alpha_i^t\|^2].$$

Proof. For ease of exposition, we use the notation

$$\zeta^t = (\nabla f_{i_t}(x^t) - \nabla f_{i_t}(\alpha_{i_t}^t))$$

Using the convexity of $\|\cdot\|^2$ and the definition of v^t we get

$$\begin{aligned}
\mathbb{E}[\|v^t\|^2] &= \mathbb{E}[\|\zeta^t + \tfrac{1}{n}\sum_{i=1}^{n} \nabla f(\alpha_i^t)\|^2] \\
&= \mathbb{E}[\|\zeta^t + \tfrac{1}{n}\sum_{i=1}^{n} \nabla f(\alpha_i^t) - \nabla f(x^t) + \nabla f(x^t)\|^2] \\
&\leq 2\mathbb{E}[\|\nabla f(x^t)\|^2] + 2\mathbb{E}[\|\zeta^t - \mathbb{E}[\zeta^t]\|^2] \\
&\leq 2\mathbb{E}[\|\nabla f(x^t)\|^2] + 2\mathbb{E}[\|\zeta^t\|^2].
\end{aligned}$$

The first inequality follows from the fact that $\|a+b\|^2 \leq 2(\|a\|^2 + \|b\|^2)$ and that $\mathbb{E}[\zeta^t] = \nabla f(x^t) - \frac{1}{n}\sum_{i=1}^{n} \nabla f(\alpha_i^t)$. The second inequality is obtained by noting that for a random variable ζ, $\mathbb{E}[\|\zeta - \mathbb{E}[\zeta]\|^2] \leq \mathbb{E}[\|\zeta\|^2]$. Using Jensen's inequality in the inequality above, we get

$$\begin{aligned}
&\mathbb{E}[\|v^t\|^2] \\
&\leq 2\mathbb{E}[\|\nabla f(x^t)\|^2] + \frac{2}{n}\sum_{i=1}^{n} \mathbb{E}[\|\nabla f_{i_t}(x^t) - \nabla f_{i_t}(\alpha_i^t)\|^2] \\
&\leq 2\mathbb{E}[\|\nabla f(x^t)\|^2] + \frac{2L^2}{n}\sum_{i=1}^{n} \mathbb{E}[\|x^t - \alpha_i^t\|^2].
\end{aligned}$$

The last inequality follows from L-smoothness of f_{i_t}, thus concluding the proof. □

The following result provides a bound on the function value of functions with Lipschitz continuous gradients.

Lemma 3.9.2. *Suppose the function $f : \mathbb{R}^d \to \mathbb{R}$ is L-smooth, then the following holds*

$$f(x) \leq f(y) + \langle \nabla f(y), x - y \rangle + \frac{L}{2}\|x - y\|^2,$$

for all $x, y \in \mathbb{R}^d$.

Chapter 4

A Generic Approach for Escaping Saddle Points

4.1 Introduction

While the previous chapters examined fast stochastic and incremental methods for nonconvex optimization, the convergence criterion was that of stationarity. In this chapter, we investigate algorithms for nonconvex *finite-sum* problems of the form

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

with focus on convergence to local minimizers. Recall that neither $f : \mathbb{R}^d \to \mathbb{R}$ nor the individual functions $f_i : \mathbb{R}^d \to \mathbb{R}$ ($i \in [n]$) are necessarily convex. As earlier, we operate in a general nonconvex setting except for few smoothness assumptions like Lipschitz continuity of the gradient and Hessian.

In the large-scale settings, algorithms based on first-order information of functions f_i are typically favored as they are relatively inexpensive and scale seamlessly. An algorithm widely used in practice is stochastic gradient descent (SGD), which has the iterative update:

$$x_{t+1} = x_t - \eta_t \nabla f_{i_t}(x_t), \tag{4.2}$$

where $i_t \in [n]$ is a randomly chosen index and η_t is a learning rate. Under suitable selection of the learning rate, we can show that SGD converges to a point x that, in expectation, satisfies the stationarity condition $\|\nabla f(x)\| \leq \epsilon$ in $O(1/\epsilon^4)$ iterations [47]. This result has two critical weaknesses: (i) It does not ensure convergence to local optima or second-order critical points; (ii) The rate of convergence of the SGD algorithm is slow.

For general nonconvex problems, one has to settle for a more modest goal than sub-optimality, as finding the global minimizer of finite-sum nonconvex problem will be in general intractably hard. Unfortunately, SGD does not even ensure second-order critical conditions such as local optimality since it can get stuck at saddle points. This issue

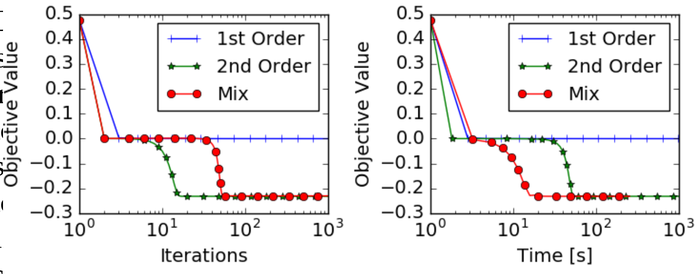


Figure 4.1: First order methods like GI can potentially get stuck at saddle points Second-order methods can escape it ir very few iterations (as observed in th left plot) but at the cost of expensive Hes sian based iterations (see time plot to th right). The proposed framework, which i a novel mix of the two strategies, can escape saddle points *faster* in time by carefully trading off computation and iteration complexity.

has recently received considerable attention in the ML community, especially in the context of deep learning [27, 29, 30]. These works argue that saddle points are highly prevalent in most optimization paths, and are the primary obstacle for training large deep networks. To tackle this issue and achieve a second-order critical point for which $\|\nabla f\| \leq \epsilon$ and $\nabla^2 f \succeq -\sqrt{\epsilon}\mathbb{I}$, we need algorithms that either use the Hessian explicitly or exploit its structure.

A key work that explicitly uses Hessians to obtain faster convergence rates is the cubic regularization (CR) method [115]. In particular, Nesterov and Polyak [115] showed that CR requires $O(1/\epsilon^{3/2})$ iterations to achieve the second-order critical conditions. However, each iteration of CR is expensive as it requires computing the Hessian and solving multiple linear systems, each of which has complexity $O(d^{\omega})$ (ω is the matrix multiplication constant), thus, undermining the benefit of its faster convergence. Recently, Agarwal et al. [5] designed an algorithm to solve the CR more efficiently, however, it still exhibits slower convergence in practice compared to first-order methods. Both of these approaches use Hessian based optimization in each iteration, which make them slow in practice.

A second line of work focuses on using Hessian information (or its structure) whenever the method gets stuck at stationary points that are not second-order critical. To our knowledge, the first work in this line is [46], which shows that for a class of functions that satisfy a special property called "strict-saddle" property, a noisy variant of SGD can converge to a point close to a local minimum. For this class of functions, points close to saddle points have a Hessian with a large negative eigenvalue, which proves instrumental in escaping saddle points using an isotropic noise. While such a noise-based method is appealing as it only uses first-order information, it has a very bad dependence on the dimension d, and furthermore, the result only holds when the strict-saddle property is satisfied [46]. More recently, Carmon et al. [22] presented a new faster algorithm that alternates between first-order and second-order subroutines. However, their algorithm is designed for the simple case of $n = 1$ in (4.1) and hence, can be expensive in practice.

Inspired by this line of work, we develop a general framework for finding second-order critical points. The key idea of our framework is to use first-order information for the most part of the optimization process and invoke Hessian information only when stuck at stationary points that are not second-order critical. We summarize the key idea and main contributions of this chapter below.

Main Contributions: We develop an algorithmic framework for converging to second-order critical points and provide convergence analysis for it. Our framework carefully alternates between two subroutines that use gradient and Hessian information, respectively, and ensures second-order criticality. Furthermore, we present two instantiations of our framework and provide convergence rates for them. In particular, we show that a simple instance of our framework, based on SVRG, achieves convergence rates competitive with the current state-of-the-art methods; thus highlighting the simplicity and applicability of our framework. Finally, we demonstrate the empirical performance of a few algorithms encapsulated by our framework and show their superior performance.

4.1.1 Related Work

There is a vast literature on algorithms for solving optimization problems of the form (4.1). A classical approach for solving such optimization problems is SGD, which dates back at least to the seminal work of [148]. Since then, SGD has been a subject of extensive research, especially in the convex setting [19, 77, 93, 124]. Recently, new faster methods, called variance reduced (VR) methods, have been proposed for convex finite-sum problems. VR methods attain faster convergence by reducing the variance in the stochastic updates of SGD, see e.g., [33, 34, 71, 75, 153, 156]. Accelerated variants of these methods achieve the lower bounds proved in [2, 81], thereby settling the question of their optimality. Furthermore, [133] developed an asynchronous framework for VR methods and demonstrated their benefits in parallel environments.

Most of the aforementioned prior works study stochastic methods in convex or very specialized nonconvex settings that admit theoretical guarantees on sub-optimality. For the general nonconvex setting, it is only recently that non-asymptotic convergence rate analysis for SGD and its variants was obtained in [47], who showed that SGD ensures $\|\nabla f\| \leq \epsilon$ (in expectation) in $O(1/\epsilon^4)$ iterations. A similar rate for parallel and distributed SGD was shown in [87]. For these problems, Reddi et al. [139, 141, 142] proved faster convergence rates that ensure the same optimality criteria in $O(n + n^{2/3}/\epsilon^2)$, which is an order $n^{1/3}$ faster than GD. While these methods ensure convergence to *stationary* points at a faster rate, the question of convergence to local minima (or in general to second-order critical points) has not been addressed. To our knowledge, convergence rates to second-order critical points (defined in Definition 4.2.1) for general nonconvex functions was first studied by [115]. However, each iteration of the algorithm in [115] is prohibitively expensive since it requires eigenvalue decompositions, and hence, is unsuitable for large-scale high-dimensional problems. More recently, Agarwal et al. [5], Carmon et al. [22] presented algorithms for finding second-order critical points by tackling some practical issues that arise in [115]. However, these algorithms are either only applicable to a restricted setting or heavily use Hessian based computations, making them unappealing from a practical standpoint. *Noisy* variants of first-order methods have also been shown to escape saddle points (see [46, 69, 83]), however, these methods have strong dependence on either n or d, both of which are undesirable.

4.2 Background & Problem Setup

We assume that each of the functions f_i in (4.1) is L-smooth, i.e., $\|\nabla f_i(x) - \nabla f_i(y)\| \leq L\|x - y\|$ for all $i \in [n]$. Furthermore, we assume that the Hessian of f in (4.1) is Lipschitz, i.e., we have

$$\|\nabla^2 f(x) - \nabla^2 f(y)\| \leq M\|x - y\|, \tag{4.3}$$

for all $x, y \in \mathbb{R}^d$. Such a condition is typically necessary to ensure convergence of algorithms to the second-order critical points [115]. In addition to the above smoothness conditions, we also assume that the function f is bounded below, i.e., $f(x) \geq B$ for all $x \in \mathbb{R}^d$.

In order to measure stationarity of an iterate x, similar to [47, 111, 115], we use the condition $\|\nabla f(x)\| \leq \epsilon$. In this chapter, we are interested in convergence to second-order critical points. Thus, in addition to stationarity, we also require the solution to satisfy the Hessian condition $\nabla^2 f(x) \succeq -\gamma\mathbb{I}$ [115]. For iterative algorithms, we require both $\epsilon, \gamma \to 0$ as the number of iterations $T \to \infty$. When all saddle points are nondegenerate, such a condition implies convergence to a local optimum.

Definition 4.2.1. *An algorithm $\mathcal{A}$ is said to obtain a point x that is a (ϵ, γ)-second order critical point if $\mathbb{E}[\|\nabla f(x)\|] \leq \epsilon$ and $\nabla^2 f(x) \succeq -\gamma\mathbb{I}$, where the expectation is over any randomness in $\mathcal{A}$.*

We must exercise caution while interpreting results pertaining to (ϵ, γ)-second order critical points. Such points need not be close to any local minima either in objective function value, or in the domain of (4.1). Note that the aforementioned criterion is stronger than the one used in the previous chapters. For our algorithms, we use only an Incremental First-order Oracle (IFO) [2] and an Incremental Second-order Oracle (ISO), defined below.

Definition 4.2.2. *An IFO takes an index $i \in [n]$ and a point $x \in \mathbb{R}^d$, and returns the pair $(f_i(x), \nabla f_i(x))$. An ISO takes an index $i \in [n]$, point $x \in \mathbb{R}^d$ and vector $v \in \mathbb{R}^d$ and returns the vector $\nabla^2 f_i(x)v$.*

IFO and ISO calls are typically cheap, with ISO call being relatively more expensive. In many practical settings that arise in machine learning, the time complexity of these oracle calls is linear in d [4, 120]. For clarity and clean comparison, the dependence of time complexity on Lipschitz constant L, M, initial point and any polylog factors in the results is hidden.

4.3 Generic Framework

In this section, we propose a generic framework for escaping saddle points while solving nonconvex problems of form (4.1). One of the primary difficulties in reaching a second-order critical point is the presence of saddle points. To evade such points, one needs to use properties of both gradients and Hessians. To this end, our framework is based on two core subroutines: GF-OPTIMIZER and HF-OPTIMIZER.

Algorithm 8: Generic Framework

```
1: Input - Initial point $x^0$, iterations $T$, error threshold parameters $\epsilon$, $\gamma$ and probability $p$
2: for $t = 1$ to $T$ do
3:     $(y^t, z^t) = \text{GF-OPTIMIZER}(x^{t-1}, \epsilon)$ (refer to G.1 and G.2)
4:     Choose $u^t$ as $y^t$ with probability $p$ and $z^t$ with probability $1 - p$
5:     $(x^{t+1}, \tau^{t+1}) = \text{HF-OPTIMIZER}(u^t, \epsilon, \gamma)$ (refer to H.1 and H.2)
6:     if $\tau^{t+1} = \emptyset$ then
7:         Output set $\{x^{t+1}\}$
8:     end if
9: end for
10: Output set $\{y^1, ..., y^T\}$
```

The idea is to use these two subroutines, each focused on different aspects of the optimization procedure. GF-OPTIMIZER focuses on using gradient information for decreasing the function. On its own, the GF-OPTIMIZER might not converge to a local minimizer since it can get stuck at a saddle point. Hence, we require the subroutine HF-OPTIMIZER to help avoid saddle points. A natural idea is to interleave these subroutines to obtain a second-order critical point. But it is not even clear if such a procedure even converges. We propose a carefully designed procedure that effectively balances these two subroutines, which not only provides meaningful theoretical guarantees, but remarkably also translates into strong empirical gains in practice.

Algorithm 8 provides pseudocode of our framework. Observe that the algorithm is still abstract, since it does not specify the subroutines GF-OPTIMIZER and HF-OPTIMIZER. These subroutines determine the crucial update mechanism of the algorithm. We will present specific instance of these subroutines in the next section, but we assume the following properties to hold for these subroutines.

- GF-OPTIMIZER: Suppose (y, z) = GF-OPTIMIZER(x, n, ϵ), then there exists positive function $g : \mathbb{N} \times \mathbb{R}^+ \to \mathbb{R}^+$, such that

  G.1 $\mathbb{E}[f(y)] \leq f(x)$,

  G.2 $\mathbb{E}[\|\nabla f(y)\|^2] \leq \frac{1}{g(n,\epsilon)} \mathbb{E}[f(x) - f(z)]$.

  Here the outputs $y, z \in \mathbb{R}^d$. The expectation in the conditions above is over any randomness that is a part of the subroutine. The function g will be critical for the overall rate of Algorithm 8. Typically, GF-OPTIMIZER is a first-order method, since the primary aim of this subroutine is to focus on gradient based optimization.

- HF-OPTIMIZER: Suppose (y, τ) = HF-OPTIMIZER(x, n, ϵ, γ) where $y \in \mathbb{R}^d$ and $\tau = \{\emptyset, \diamond\}$. If $\tau = \emptyset$, then y is a (ϵ, γ)-second order critical point with probability at least $1 - q$. Otherwise if $\tau = \diamond$, then y satisfies the following condition:

  H.1 $\mathbb{E}[f(y)] \leq f(x)$,

  H.2 $\mathbb{E}[f(y)] \leq f(x) - h(n, \epsilon, \gamma)$ when $\lambda_{\min}(\nabla^2 f(x)) \leq -\gamma$ for some function $h : \mathbb{N} \times \mathbb{R}^+ \times \mathbb{R}^+ \to \mathbb{R}^+$.

Here the expectation is over any randomness in subroutine HF-OPTIMIZER. The two conditions ensure that the objective function value, in expectation, never increases and furthermore, decreases with a certain rate when $\lambda_{\min}(\nabla^2 f(x)) \leq -\gamma$. In general, this subroutine utilizes the Hessian or its properties for minimizing the objective function. Typically, this is the most expensive part of the Algorithm 8 and hence, needs to be invoked judiciously.

The key aspect of these subroutines is that they, in expectation, never increase the objective function value. The functions g and h will determine the convergence rate of Algorithm 8. In order to provide a concrete implementation, we need to specify the aforementioned subroutines. Before we delve into those details, we will provide a generic convergence analysis for Algorithm 8.

Convergence Analysis

Theorem 4.3.1. *Let $\Delta = f(x^0) - B$ and $\theta = \min((1-p)\epsilon^2 g(n,\epsilon), ph(n,\epsilon,\gamma))$. Also, let set Γ be the output of Algorithm 8 with* GF-OPTIMIZER *satisfying **G.1** and **G.2** and* HF-OPTIMIZER *satisfying **H.1** and **H.2**. Furthermore, T be such that $T > \Delta/\theta$.*

Suppose the multiset $S = \{i_1, ...i_k\}$ are k indices selected independently and uniformly randomly from $\{1, ..., |\Gamma|\}$. Then the following holds for the indices in S:

1. *y^t, where $t \in \{i_1, ..., i_k\}$, is a (ϵ, γ)-critical point with probability at least $1 - \max(\Delta/(T\theta), q)$.*
2. *If $k = O(\log(1/\zeta)/\min(\log(\Delta/(T\theta)), \log(1/q)))$, with at least probability $1 - \zeta$, at least one iterate y^t where $t \in \{i_1, ..., i_k\}$ is a (ϵ, γ)-critical point.*

The proof of the result is presented in Appendix 4.7. The key point regarding the above result is that the overall convergence rate depends on the magnitude of both functions g and h. Theorem 4.3.1 shows that the slowest amongst the subroutines GF-OPTIMIZER and HF-OPTIMIZER governs the overall rate of Algorithm 8. Thus, it is important to ensure that both these procedures have good convergence. Also, note that the optimal setting for p based on the result above satisfies $1/p = 1/\epsilon^2 g(n,\epsilon) + 1/h(n,\epsilon,\gamma)$. We defer further discussion of convergence to next section, where we present more specific convergence and rate analysis.

4.4 Concrete Instantiations

We now present specific instantiations of our framework in this section. Before we state our key results, we discuss an important subroutine that is used as GF-OPTIMIZER for rest of this chapter: SVRG. We give a brief description of the algorithm in this section and show that it meets the conditions required for a GF-OPTIMIZER. SVRG [71, 139] is a stochastic algorithm recently shown to be very effective for reducing variance in finite-sum problems. We seek to understand its benefits for nonconvex optimization, with a particular focus on the issue of escaping saddle points. Algorithm 9 presents SVRG's pseudocode.

Algorithm 9: SVRG(x^0, ϵ)

```
1: Input: $x_m^0 = x^0 \in \mathbb{R}^d$, epoch length $m$, step sizes $\{\eta_i > 0\}_{i=0}^{m-1}$, iterations $T_g$, $S = \lceil T_g/m \rceil$
2: for $s = 0$ to $S-1$ do
3:     $\tilde{x}^s = x_0^{s+1} = x_m^s$
4:     $g^{s+1} = \frac{1}{n}\sum_{i=1}^n \nabla f_i(\tilde{x}^s)$
5:     for $t = 0$ to $m-1$ do
6:         Uniformly randomly pick $i_t$ from $\{1, \dots, n\}$
7:         $v_t^{s+1} = \nabla f_{i_t}(x_t^{s+1}) - \nabla f_{i_t}(\tilde{x}^s) + g^{s+1}$
8:         $x_{t+1}^{s+1} = x_t^{s+1} - \eta_t v_t^{s+1}$
9:     end for
10: end for
11: Output: $(y, z)$ where $y$ is Iterate $x_a$ chosen uniformly random from $\{\{x_t^{s+1}\}_{t=0}^{m-1}\}_{s=0}^{S-1}$ and
    $z = x_m^S$.
```

Observe that Algorithm 9 is an epoch-based algorithm. At the start of each epoch s, a full gradient is calculated at the point $\tilde{x}^s$, requiring n calls to the IFO. Within its inner loop SVRG performs m stochastic updates. Suppose m is chosen to be $O(n)$ (typically used in practice), then the total IFO calls per epoch is $\Theta(n)$. Strong convergence rates have been proved Algorithm 9 in the context of convex and nonconvex optimization [71, 139]. The following result shows that SVRG meets the requirements of a GF-OPTIMIZER.

Lemma 4.4.1. *Suppose $\eta_t = \eta = 1/4Ln^{2/3}$, $m = n$ and $T_g = T_\epsilon$, which depends on ϵ, then Algorithm 9 is a* GF-OPTIMIZER *with $g(n, \epsilon) = T_\epsilon/40Ln^{2/3}$.*

In rest of this section, we discuss approaches using SVRG as a GF-OPTIMIZER. In particular, we propose and provide convergence analysis for two different methods with different HF-OPTIMIZER but which use SVRG as a GF-OPTIMIZER.

4.4.1 Hessian descent

The first approach is based on directly using the eigenvector corresponding to the smallest eigenvalue as a HF-OPTIMIZER. More specifically, when the smallest eigenvalue of the Hessian is negative and reasonably large in magnitude, the Hessian information can be used to ensure descent in the objective function value. The pseudo-code for the algorithm is given in Algorithm 10.

The key idea is to utilize the minimum eigenvalue information in order to make a descent step. If $\lambda_{\min}(\nabla^2 f(x)) \leq -\gamma$ then the idea is to use this information to take a descent step. Note the subroutine is designed in a fashion such that the objective function value never increases. Thus, it naturally satisfies the requirement **H.1** of HF-OPTIMIZER. The following result shows that HESSIANDESCENT is a HF-OPTIMIZER.

Lemma 4.4.2. HESSIANDESCENT *is a* HF-OPTIMIZER *with $h(n, \epsilon, \gamma) = \frac{\rho}{24M^2}\gamma^3$.*

The proof of the result is presented in Appendix 4.9. With SVRG as GF-OPTIMIZER and HESSIANDESCENT as HF-OPTIMIZER, we show the following key result:

Algorithm 10: HESSIANDESCENT (x, ϵ, γ)

```
1: Find $v$ such that $\|v\| = 1$, and with probability at least $\rho$ the following inequality holds:
   $\langle v, \nabla^2 f(x) v\rangle \leq \lambda_{min}(\nabla^2 f(x)) + \frac{\gamma}{2}$.
2: Set $\alpha = |\langle v, \nabla^2 f(x) v\rangle| / M$.
3: $u = x - \alpha \, \text{sign}(\langle v, \nabla f(x)\rangle) v$.
4: $y = \arg\min_{z \in \{u, x\}} f(z)$
5: Output: $(y, \diamond)$.
```

Theorem 4.4.3. *Suppose* SVRG *with $m = n$, $\eta_t = \eta = 1/4Ln^{2/3}$ for all $t \in \{1, ..., m\}$ and $T_g = 40Ln^{2/3}/\epsilon^{1/2}$ is used as* GF-OPTIMIZER *and* HESSIANDESCENT *is used as* HF-OPTIMIZER *with $q = 0$, then Algorithm 8 finds a $(\epsilon, \sqrt{\epsilon})$-second order critical point in $T = O(\Delta / \min(p, 1-p)\epsilon^{3/2})$ with probability at least* 0.9.

The result directly follows from using Lemma 4.4.1 and 4.4.2 in Theorem 4.3.1. The result shows that the iteration complexity of Algorithm 8 in this case is $O(\Delta/\epsilon^{3/2} \min(p, 1-p))$. Thus, the overall IFO complexity of SVRG algorithm is $(n + T_g) \times T = O(n/\epsilon^{3/2} + n^{2/3}/\epsilon^2)$. Since each IFO call takes $O(d)$ time, the overall time complexity of all GF-OPTIMIZER steps is $O(nd/\epsilon^{3/2} + n^{2/3}d/\epsilon^2)$. To understand the time complexity of HESSIANDESCENT, we need the following result [5].

Preposition 1. *The time complexity of finding $v \in \mathbb{R}^d$ that $\|v\| = 1$, and with probability at least ρ the following inequality holds: $\langle v, \nabla^2 f(x) v\rangle \leq \lambda_{min}(\nabla^2 f(x)) + \frac{\gamma}{2}$ is $O(nd + n^{3/4}d\gamma^{1/2})$.*

Note that each iteration of Algorithm 8 in this case has just linear dependence on d. Since the total number of HESSIANDESCENT iterations is $O(\Delta/\min(p, 1-p)\epsilon^{3/2})$ and each iteration has the complexity of $O(nd + n^{3/4}d/\epsilon^{1/4})$, using the above remark, we obtain an overall time complexity of HESSIANDESCENT is $O(nd/\epsilon^{3/2} + n^{3/4}d/\epsilon^{7/4})$. Combining this with the time complexity of SVRG, we get the following result.

Corollary 4.4.3.1. *The overall running time of Algorithm 8 to find a $(\epsilon, \sqrt{\epsilon})$-second order critical point, with parameter settings used in Theorem 4.4.3, is $O(nd/\epsilon^{3/2} + n^{3/4}d/\epsilon^{7/4} + n^{2/3}d/\epsilon^2)$.*

Note that the dependence on ϵ is much better in comparison to that of Noisy SGD used in [46]. Furthermore, our results are competitive with [5, 22] in their respective settings, but with a much simpler algorithm and analysis. We also note that our algorithm is faster than the one proposed in [69], which has a time complexity of $O(nd/\epsilon^2)$.

4.4.2 Cubic Descent

In this section, we show that the cubic regularization method in [115] can be used as HF-OPTIMIZER. More specifically, here HF-OPTIMIZER approximately solves the following optimization problem:

$$y = \arg\min_z \langle \nabla f(x), z - x\rangle + \frac{1}{2}\left\langle z - x, \nabla^2 f(x)(z - x)\right\rangle + \frac{M}{6}\|z - x\|^3, \tag{CUBICDESCENT}$$

and returns $(y, \diamond)$ as output. The following result can be proved for this approach.

Theorem 4.4.4. *Suppose* SVRG *(with same parameters as in Theorem 4.4.3) is used as* GF-OPTIMIZER *and* CUBICDESCENT *is used as* HF-OPTIMIZER *with $q = 0$, then Algorithm 8 finds a $(\epsilon, \sqrt{\epsilon})$-second order critical point in $T = O(\Delta / \min(p, 1-p)\epsilon^{3/2})$ with probability at least* 0.9.

In principle, Algorithm 8 with CUBICDESCENT as HF-OPTIMIZER can converge without the use of GF-OPTIMIZER subroutine at each iteration since it essentially reduces to the cubic regularization method of [115]. However, in practice, we would expect GF-OPTIMIZER to perform most of the optimization and HF-OPTIMIZER to be used for far fewer iterations. Using the method developed in [115] for solving CUBICDESCENT, we obtain the following corollary.

Corollary 4.4.4.1. *The overall running time of Algorithm 8 to find a $(\epsilon, \sqrt{\epsilon})$-second order critical point, with parameter settings used in Theorem 4.4.4, is $O(nd^{\omega}/\epsilon^{3/2} + n^{2/3}d/\epsilon^2)$.*

Here ω is the matrix multiplication constant. The dependence on ϵ is weaker in comparison to Corollary 4.4.3.1. However, each iteration of CUBICDESCENT is expensive (as seen from the factor d^{ω} in the corollary above) and thus, in high dimensional settings typically encountered in machine learning, this approach can be expensive in comparison to HESSIANDESCENT.

4.4.3 Practical Considerations

The focus of this section was to demonstrate the wide applicability of our framework; wherein using a simple instantiation of this framework, we could achieve algorithms with fast convergence rates. To further achieve good empirical performance, we had to slightly modify these procedures. For HF-OPTIMIZER, we found stochastic, adaptive and inexact approaches for solving HESSIANDESCENT and CUBICDESCENT work well in practice. The exact description of these modifications is deferred to Appendix 4.12. Furthermore, in the context of deep learning, empirical evidence suggests that first-order methods like ADAM [73] exhibit behavior that is in congruence with properties **G.1** and **G.2**. While theoretical analysis for a setting where ADAM is used as GF-OPTIMIZER is still unresolved, we nevertheless demonstrate its performance through empirical results in the following section.

4.5 Experiments

We now present empirical results for our saddle point avoidance technique with an aim to highlight three aspects: (i) the framework successfully escapes non-degenerate saddle points, (ii) the framework is fast, and (iii) the framework is practical on large-scale problems. All the algorithms are implemented on TensorFlow [1]. In case of deep networks, the Hessian-vector product is evaluated using the trick presented in [120]. We run our experiments on a commodity machine with Intel® Xeon® CPU E5-2630 v4 CPU, 256GB RAM, and NVidia® Titan X (Pascal) GPU.

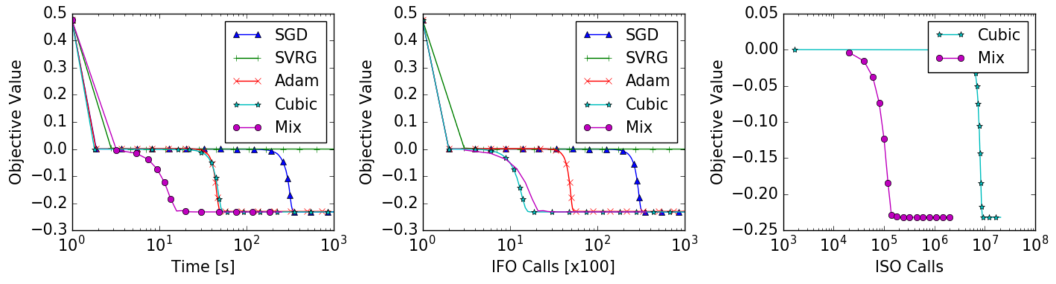


Figure 4.2: Comparison of various methods on a synthetic problem. Our mix framework successfully escapes saddle point and uses relatively few ISO calls in comparison to CUBICDESCENT.

Synthetic Problem To demonstrate the fast escape from a saddle point by the proposed method, we consider the following simple nonconvex finite-sum problem:

$$\min_{x \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} x^T A_i x + b_i^T x + \|x\|_{10}^{10} \tag{4.4}$$

Here the parameters are designed such that $\sum_i b_i = 0$ and $\sum_i A_i$ matrix has exactly one negative eigenvalue of -0.001 and other eigenvalues randomly chosen in the interval $[1, 2]$. The total number of examples n is set to be 100,000 and d is 1000. It is not hard to see that this problem has a non-degenerate saddle point at the origin. This allows us to explore the behaviour of different optimization algorithms in the vicinity of the saddle point. In this experiment, we compare a mix of SVRG and HESSIANDESCENT (as in Theorem 4.4.3) with SGD (with constant step size), ADAM, SVRG and CUBICDESCENT. The parameter of these algorithms is chosen by grid search so that it gives the best performance. The subproblem of CUBICDESCENT was solved with gradient descent [22] until the gradient norm of the subproblem is reduced below 10^{-3}. We study the progress of optimization, i.e., decrease in function value with wall clock time, IFO calls, and ISO calls. All algorithms were initialized with the same starting point very close to origin.

The results are presented in Figure 4.2, which shows that our proposed mix framework was the *fastest* to escape the saddle point in terms of wall clock time. We observe that performance of the first order methods suffered severely due to the saddle point. Note that SGD eventually escaped the saddle point due to inherent noise in the mini-batch gradient. CUBICDESCENT, a second-order method, escaped the saddle point faster in terms of iterations using the Hessian information. But operating on Hessian information is expensive as a result this method was slow in terms of wall clock time. The proposed framework, which is a mix of the two strategies, inherits the best of both worlds by using cheap gradient information most of the time and reducing the use of relatively expensive Hessian information (ISO calls) by 100x. This resulted in *faster* escape from saddle point in terms of wall clock time.

Deep Networks To investigate the practical performance of the framework for deep learning problems, we applied it to two deep autoencoder optimization problems from

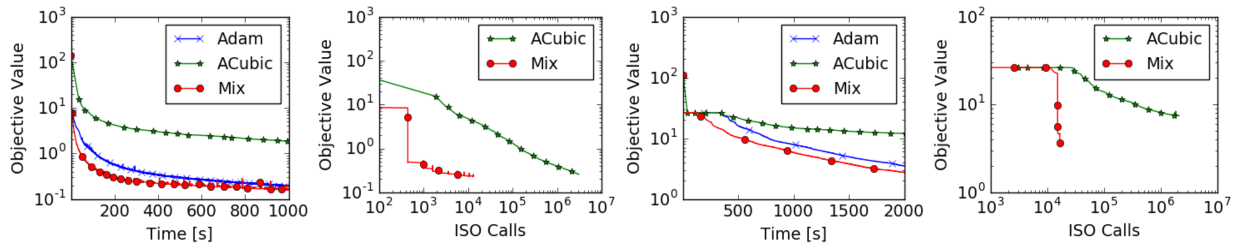


Figure 4.3: Comparison of various methods on CURVES and MNIST Deep Autoencoder. Our mix approach converges faster than the baseline methods and uses relatively few ISO calls in comparison to APPROXCUBICDESCENT.

[59] called "CURVES" and "MNIST". Due to their high difficulty, performance on these problems has become a standard benchmark for neural network optimization methods, e.g. [100, 101, 165, 170]. The "CURVES" autoencoder consists of an encoder with layers of size (28x28)-400-200-100- 50-25-6 and a symmetric decoder totaling in 0.85M parameters. The six units in the code layer were linear and all the other units were logistic. The network was trained on 20,000 images and tested on 10,000 new images. The data set contains images of curves that were generated from three randomly chosen points in two dimensions. The "MNIST" autoencoder consists of an encoder with layers of size (28x28)-1000-500-250-30 and a symmetric decoder, totaling in 2.8M parameters. The thirty units in the code layer were linear and all the other units were logistic. The network was trained on 60,000 images and tested on 10,000 new images. The data set contains images of handwritten digits 0-9. The pixel intensities were normalized to lie between 0 and 1.[1]

As an instantiation of our framework, we use a mix of ADAM, which is popular in deep learning community, and an APPROXCUBICDESCENT for the practical reasons mentioned in Section 4.4.3. This method with ADAM and APPROXCUBICDESCENT. The parameters of these algorithms were chosen to produce the best generalization on a held out test set. The regularization parameter M was chosen as the smallest value such that the function value does not fluctuate in the first 10 epochs. We use the initialization suggested in [100] and a mini-batch size of 1000 for all the algorithms. We report objective function value against wall clock time and ISO calls.

The results are presented in Figure 4.3, which shows that our proposed mix framework was the *fastest* to escape the saddle point in terms of wall clock time. ADAM took considerably more time to escape the saddle point, especially in the case of MNIST. While APPROXCUBICDESCENT escaped the saddle point in relatively fewer iterations, each iteration required considerably large number of ISO calls; as a result, the method was extremely slow in terms of wall clock time, despite our efforts to improve it via approximations and code optimizations. On the other hand, our proposed framework, seamlessly balances these two methods, thereby, resulting in the fast decrease of training loss.

[1]Data available at: www.cs.toronto.edu/~jmartens/digs3pts_1.mat, mnist_all.mat

4.6 Discussion

In this chapter, we examined a generic strategy to escape saddle points in nonconvex finite-sum problems and presented its convergence analysis. The key intuition is to alternate between a first-order and second-order based optimizers; the latter is mainly intended to escape points that are only stationary but are not second-order critical points. We presented two different instantiations of our framework and provided their detailed convergence analysis. While both our methods explicity use the Hessian information, one can also use noisy first-order methods as HF-OPTIMIZER (see e.g. noisy SGD in [46]). In such a scenario, we exploit the negative eigenvalues of the Hessian to escape saddle points by using isotropic noise, and do not explicitly use ISO. For these methods, under strict-saddle point property [46], we can show convergence to local optima within our framework.

We primarily focused on obtaining second-order critical points for nonconvex finite-sums (4.1). This does not necessarily imply low test error or good generalization capabilities. Thus, we should be careful when interpreting the results presented in this chapter. A detailed discussion or analysis of these issues is out of scope of this thesis. While a few prior works argue for convergence to local optima, the exact connection between generalization and local optima is not well understood, and is an interesting open problem. Nevertheless, we believe the techniques presented in this chapter can be used alongside other optimization tools for faster and better nonconvex optimization.

Appendix: Omitted Proofs and Additional Experiments

4.7 Proof of Theorem 4.3.1

The case of $\tau = \emptyset$ can be handled in a straightforward manner, so let us focus on the case where $\tau = \diamond$. We split our analysis into cases, each analyzing the change in objective function value depending on second-order criticality of y^t.

We start with the case where the gradient condition of second-order critical point is violated and then proceed to the case where the Hessian condition is violated.

Case I: $\mathbb{E}[\|\nabla f(y^t)\|] \geq \epsilon$ for some $t > 0$ We first observe the following: $\mathbb{E}[\|\nabla f(y^t)\|^2] \geq (\mathbb{E}\|\nabla f(y^t)\|)^2 \geq \epsilon^2$. This follows from a straightforward application of Jensen's inequality. From this inequality, we have the following:

$$\epsilon^2 \leq \mathbb{E}[\|\nabla f(y^t)\|^2] \leq \frac{1}{g(n,\epsilon)}\mathbb{E}[f(x^{t-1}) - f(z^t)]. \tag{4.5}$$

This follows from the fact that y^t is the output of GF-OPTIMIZER subroutine, which satisfies the condition that for (y, z) = GF-OPTIMIZER(x, n, ϵ), we have

$$\mathbb{E}[\|\nabla f(y)\|^2] \leq \frac{1}{g(n,\epsilon)}\mathbb{E}[f(x) - f(z)].$$

From Equation (4.5), we have

$$\mathbb{E}[f(z^t)] \leq \mathbb{E}[f(x^{t-1})] - \epsilon^2 g(n,\epsilon).$$

Furthermore, due to the property of non-increasing nature of GF-OPTIMIZER, we also have $\mathbb{E}[y^t] \leq \mathbb{E}[f(x^{t-1})]$.

We now focus on the HF-OPTIMIZER subroutine. From the property of HF-OPTIMIZER that the objective function value is non-increasing, we have $\mathbb{E}[f(x^t)] \leq \mathbb{E}[f(u^t)]$. Therefore, combining with the above inequality, we have

$$\begin{aligned}\mathbb{E}[f(x^t)] &\leq \mathbb{E}[f(u^t)] \\ &= p\mathbb{E}[f(y^t)] + (1-p)\mathbb{E}[f(z^t)] \\ &\leq p\mathbb{E}[f(x^{t-1})] + (1-p)(\mathbb{E}[f(x^{t-1})] - \epsilon^2 g(n,\epsilon)) \\ &= \mathbb{E}[f(x^{t-1})] - (1-p)\epsilon^2 g(n,\epsilon). \end{aligned} \tag{4.6}$$

The first equality is due to the definition of u^t in Algorithm 8. Therefore, when the gradient condition is violated, irrespective of whether $\lambda_{\min}(\nabla^2 f(x)) \leq -\gamma$ or $\nabla^2 f(y^t) \succeq -\gamma\mathbb{I}$, the objective function value always decreases by at least $\epsilon^2 g(n,\epsilon)$.
Case II: $\mathbb{E}[\|\nabla f(y^t)\|] < \epsilon$ and $\lambda_{\min}(\nabla^2 f(x)) \leq -\gamma$ for some $t > 0$ In this case, we first note that for $y =$ HF-OPTIMIZER(x, n, ϵ, γ) and $\lambda_{\min}(\nabla^2 f(x)) \leq -\gamma$, we have $\mathbb{E}[f(y)] \leq f(x) - h(n,\epsilon,\gamma)$. Observe that $x^t =$ HF-OPTIMIZER$(u^t, n, \epsilon, \gamma)$. Therefore, if $u^t = y^t$ and $\lambda_{\min}(\nabla^2 f(x)) \leq -\gamma$, then we have

$$\mathbb{E}[f(x^t)|u^t = y^t] \leq f(y^t) - h(n,\epsilon,\gamma) \leq f(x^{t-1}) - h(n,\epsilon,\gamma).$$

The second inequality is due to the non-increasing property of GF-OPTIMIZER. On the other hand, if $u^t = z^t$, we have hand, if we have $\mathbb{E}[f(x^t)|u^t = z^t] \leq f(z^t)$. This is due to the non-increasing property of HF-OPTIMIZER. Combining the above two inequalities and using the law of total expectation, we get

$$\begin{aligned}\mathbb{E}[f(x^t)] &= p\mathbb{E}[f(x^t)|u^t = y^t] + (1-p)\mathbb{E}[f(x^t)|u^t = z^t] \\ &\leq p\left(\mathbb{E}[f(y^t)] - h(n,\epsilon,\gamma)\right) + (1-p)\mathbb{E}[f(z^t)] \\ &\leq p\Big(\mathbb{E}[f(x^{t-1})] - h(n,\epsilon,\gamma)\Big) + (1-p)\mathbb{E}[f(x^{t-1})] \\ &= \mathbb{E}[f(x^{t-1})] - ph(n,\epsilon,\gamma). \end{aligned} \tag{4.7}$$

The second inequality is due to he non-increasing property of GF-OPTIMIZER. Therefore, when the hessian condition is violated, the objective function value always decreases by at least $ph(n,\epsilon,\gamma)$.
Case III: $\mathbb{E}[\|\nabla f(y^t)\|] < \epsilon$ and $\nabla^2 f(y^t) \succeq -\gamma\mathbb{I}$ for some $t > 0$ This is the favorable case for the algorithm. The only condition to note is that the objective function value will be non-increasing in this case too. This is, again, due to the non-increasing properties of subroutines GF-OPTIMIZER and HF-OPTIMIZER. In general, greater the occurrence of this case during the course of the algorithm, higher will the probability that the output of our algorithm satisfies the desired property.

The key observation is that Case I & II cannot occur large number of times since each of these cases strictly decreases the objective function value. In particular, from Equation (4.6) and (4.7), it is easy to see that each occurrence of Case I & II the following holds:

$$\mathbb{E}[f(x^t)] \leq \mathbb{E}[f(x^{t-1})] - \theta,$$

where $\theta = \min((1-p)\epsilon^2 g(n,\epsilon), ph(n,\epsilon,\gamma))$. Furthermore, the function f is lower bounded by B, thus, Case I & II cannot occur more than $(f(x^0) - B)/\theta$ times. Therefore, the probability of occurrence of Case III is at least $1 - (f(x^0) - B)/(T\theta)$, which completes the first part of the proof.

The second part of the proof simply follows from first part. As seen above, the probability of Case I & II is at most $(f(x^0) - B)/T\theta$. Therefore, probability that an element of the set S falls in Case III is at least $1 - ((f(x^0) - B)/T\theta)^k$, which gives us the required result for the second part.

4.8 Proof of Lemma 4.4.1

Proof. The proof follows from the analysis in [139] with some additional reasoning. We need to show two properties: **G.1** and **G.2**, both of which are based on objective function value. To this end, we start with an update in the s^{th} epoch. We have the following:

$$\begin{aligned}\mathbb{E}[f(x_{t+1}^{s+1})] &\leq \mathbb{E}[f(x_t^{s+1}) + \langle \nabla f(x_t^{s+1}), x_{t+1}^{s+1} - x_t^{s+1}\rangle + \tfrac{L}{2}\|x_{t+1}^{s+1} - x_t^{s+1}\|^2] \\ &\leq \mathbb{E}[f(x_t^{s+1}) - \eta_t\|\nabla f(x_t^{s+1})\|^2 + \tfrac{L\eta_t^2}{2}\|v_t^{s+1}\|^2]. \qquad (4.8)\end{aligned}$$

The first inequality is due to L-smoothness of the function f . The second inequality simply follows from the unbiasedness of SVRG update in Algorithm 9. For the analysis of the algorithm, we need the following Lyapunov function:

$$A_t^{s+1} := \mathbb{E}[f(x_t^{s+1}) + \mu_t\|x_t^{s+1} - \tilde{x}^s\|^2].$$

This function is a combination of objective function and the distance of the current iterate from the latest snapshot $\tilde{x}_s$. Note that the term μ_t is introduced only for the analysis and is not part of the algorithm (see Algorithm 9). Here $\{\mu_t\}_{t=0}^m$ is chosen such the following holds:

$$\mu_t = \mu_{t+1}(1 + \eta_t\beta_t + 2\eta_t^2 L^2) + \eta_t^2 L^3,$$

for all $t \in \{0, \cdots, m-1\}$ and $\mu_m = 0$. For bounding the Lypunov function A, we need the following bound on the distance of the current iterate from the latest snapshot:

$$\begin{aligned}&\mathbb{E}[\|x_{t+1}^{s+1} - \tilde{x}^s\|^2] = \mathbb{E}[\|x_{t+1}^{s+1} - x_t^{s+1} + x_t^{s+1} - \tilde{x}^s\|^2] \\ &= \mathbb{E}[\|x_{t+1}^{s+1} - x_t^{s+1}\|^2 + \|x_t^{s+1} - \tilde{x}^s\|^2 + 2\langle x_{t+1}^{s+1} - x_t^{s+1}, x_t^{s+1} - \tilde{x}^s\rangle] \\ &= \mathbb{E}[\eta_t^2\|v_t^{s+1}\|^2 + \|x_t^{s+1} - \tilde{x}^s\|^2] - 2\eta_t\mathbb{E}[\langle \nabla f(x_t^{s+1}), x_t^{s+1} - \tilde{x}^s\rangle] \\ &\leq \mathbb{E}[\eta_t^2\|v_t^{s+1}\|^2 + \|x_t^{s+1} - \tilde{x}^s\|^2] + 2\eta_t\mathbb{E}\left[\tfrac{1}{2\beta_t}\|\nabla f(x_t^{s+1})\|^2 + \tfrac{1}{2}\beta_t\|x_t^{s+1} - \tilde{x}^s\|^2\right]. \qquad (4.9)\end{aligned}$$

The second equality is due to the unbiasedness of the update of SVRG. The last inequality follows from a simple application of Cauchy-Schwarz and Young's inequality. Substituting Equation (4.8) and Equation (4.9) into the Lypunov function A_{t+1}^{s+1}, we obtain the following:

$$\begin{aligned}
A_{t+1}^{s+1} &\leq \mathbb{E}[f(x_t^{s+1}) - \eta_t\|\nabla f(x_t^{s+1})\|^2 + \tfrac{L\eta_t^2}{2}\|v_t^{s+1}\|^2] \\
&\quad + \mathbb{E}[\mu_{t+1}\eta_t^2\|v_t^{s+1}\|^2 + \mu_{t+1}\|x_t^{s+1} - \tilde{x}^s\|^2] \\
&\quad + 2\mu_{t+1}\eta_t\mathbb{E}\left[\tfrac{1}{2\beta_t}\|\nabla f(x_t^{s+1})\|^2 + \tfrac{1}{2}\beta_t\|x_t^{s+1} - \tilde{x}^s\|^2\right] \\
&\leq \mathbb{E}[f(x_t^{s+1}) - \left(\eta_t - \tfrac{\mu_{t+1}\eta_t}{\beta_t}\right)\|\nabla f(x_t^{s+1})\|^2 \\
&\quad + \left(\tfrac{L\eta_t^2}{2} + \mu_{t+1}\eta_t^2\right)\mathbb{E}[\|v_t^{s+1}\|^2] + (\mu_{t+1} + \mu_{t+1}\eta_t\beta_t)\,\mathbb{E}\left[\|x_t^{s+1} - \tilde{x}^s\|^2\right]. \qquad (4.10)
\end{aligned}$$

To further bound this quantity, we use Lemma 4.11.1 to bound $\mathbb{E}[\|v_t^{s+1}\|^2]$, so that upon substituting it in Equation (4.10), we see that

$$\begin{aligned}
A_{t+1}^{s+1} &\leq \mathbb{E}[f(x_t^{s+1})] - \left(\eta_t - \tfrac{\mu_{t+1}\eta_t}{\beta_t} - \eta_t^2 L - 2\mu_{t+1}\eta_t^2\right)\mathbb{E}[\|\nabla f(x_t^{s+1})\|^2] \\
&\quad + \left[\mu_{t+1}(1 + \eta_t\beta_t + 2\eta_t^2 L^2) + \eta_t^2 L^3\right]\mathbb{E}\left[\|x_t^{s+1} - \tilde{x}^s\|^2\right] \\
&\leq A_t^{s+1} - (\eta_t - \tfrac{\mu_{t+1}\eta_t}{\beta_t} - \eta_t^2 L - 2\mu_{t+1}\eta_t^2)\mathbb{E}[\|\nabla f(x_t^{s+1})\|^2].
\end{aligned}$$

The second inequality follows from the definition of μ_t and A_t^{s+1}. Since $\eta_t = \eta = 1/(4Ln^{2/3})$ for $j > 0$ and $t \in \{0, \ldots, j-1\}$,

$$A_j^{s+1} \leq A_0^{s+1} - v_n \textstyle\sum_{t=0}^{j-1}\mathbb{E}[\|\nabla f(x_t^{s+1})\|^2], \qquad (4.11)$$

where

$$v_n = (\eta_t - \tfrac{\mu_{t+1}\eta_t}{\beta_t} - \eta_t^2 L - 2\mu_{t+1}\eta_t^2).$$

We will prove that for the given parameter setting $v_n > 0$ (see the proof below). With $v_n > 0$, it is easy to see that $A_j^{s+1} \leq A_0^{s+1}$. Furthermore, note that $A_0^{s+1} = \mathbb{E}[f(x_0^{s+1}) + \mu_0\|x_0^{s+1} - \tilde{x}^s\|^2] = \mathbb{E}[f(x_0^{s+1})]$ since $x_0^{s+1} = \tilde{x}^s$ (see Algorithm 9). Also, we have

$$\mathbb{E}[f(x_j^{s+1}) + \mu_j\|x_j^{s+1} - \tilde{x}^s\|^2] \leq \mathbb{E}[f(x_0^{s+1})]$$

and thus, we obtain $\mathbb{E}[f(x_j^{s+1})] \leq \mathbb{E}[f(x_0^{s+1})]$ for all $j \in \{0, ..., m\}$. Furthermore, using simple induction and the fact that $x_0^{s+1} = x_m^s$ for all epoch $s \in \{0, ..., S-1\}$, it easy to see that $\mathbb{E}[f(x_j^{s+1})] \leq f(x^0)$. Therefore, with the definition of y specified in the output of Algorithm 9, we see that the condition **G.1** of GF-OPTIMIZER is satisfied for SVRG algorithm.

We now prove that $v_n > 0$ and also **G.2** of GF-OPTIMIZER is satisifed for SVRG algorithm. By using telescoping the sum with $j = m$ in Equation (4.11), we obtain

$$\textstyle\sum_{t=0}^{m-1}\mathbb{E}[\|\nabla f(x_t^{s+1})\|^2] \leq \dfrac{A_0^{s+1} - A_m^{s+1}}{v_n}.$$

This inequality in turn implies that

$$\textstyle\sum_{t=0}^{m-1} \mathbb{E}[\|\nabla f(x_t^{s+1})\|^2] \leq \frac{\mathbb{E}[f(\tilde{x}^s) - f(\tilde{x}^{s+1})]}{v_n}, \tag{4.12}$$

where we used that $A_m^{s+1} = \mathbb{E}[f(x_m^{s+1})] = \mathbb{E}[f(\tilde{x}^{s+1})]$ (since $\mu_m = 0$), and that $A_0^{s+1} = \mathbb{E}[f(\tilde{x}^s)]$ (since $x_0^{s+1} = \tilde{x}^s$). Now sum over all epochs to obtain

$$\frac{1}{T_g} \sum_{s=0}^{S-1} \sum_{t=0}^{m-1} \mathbb{E}[\|\nabla f(x_t^{s+1})\|^2] \leq \frac{\mathbb{E}[f(x^0) - f(x_m^S)]}{T_g v_n}. \tag{4.13}$$

Here we used the the fact that $\tilde{x}^0 = x^0$. To obtain a handle on v_n and complete our analysis, we will require an upper bound on μ_0. We observe that $\mu_0 = \frac{L}{16n^{4/3}} \frac{(1+\theta)^m - 1}{\theta}$ where $\theta = 2\eta^2 L^2 + \eta\beta$. This is obtained using the relation $\mu_t = \mu_{t+1}(1 + \eta\beta + 2\eta^2 L^2) + \eta^2 L^3$ and the fact that $\mu_m = 0$. Using the specified values of β and η we have

$$\theta = 2\eta^2 L^2 + \eta\beta = \frac{1}{8n^{4/3}} + \frac{1}{4n} \leq \frac{3}{4n}.$$

Using the above bound on θ, we get

$$\begin{aligned} \mu_0 &= \frac{L}{16n^{4/3}} \frac{(1+\theta)^m - 1}{\theta} = \frac{L((1+\theta)^m - 1)}{2(1 + 2n^{1/3})} \\ &\leq \frac{L((1 + \frac{3}{4n})^{\lfloor 4n/3 \rfloor} - 1)}{2(1 + 2n^{1/3})} \leq n^{-1/3}(L(e-1)/4), \end{aligned} \tag{4.14}$$

wherein the second inequality follows upon noting that $(1 + \frac{1}{l})^l$ is increasing for $l > 0$ and $\lim_{l\to\infty}(1 + \frac{1}{l})^l = e$ (here e is the Euler's number). Now we can lower bound v_n, as

$$v_n = \min_t \left(\eta - \tfrac{\mu_{t+1}\eta}{\beta} - \eta^2 L - 2\mu_{t+1}\eta^2\right) \geq \left(\eta - \tfrac{\mu_0 \eta}{\beta} - \eta^2 L - 2\mu_0\eta^2\right) \geq \frac{1}{40Ln^{2/3}}.$$

The first inequality holds since μ_t decreases with t. The second inequality holds since (a) μ_0/β can be upper bounded by $(e-1)/4$ (follows from Equation (4.14)), (b) $\eta^2 L \leq \eta/4$ and (c) $2\mu_0\eta^2 \leq (e-1)\eta/8$ (follows from Equation (4.14)). Substituting the above lower bound in Equation (4.13), we obtain the following:

$$\frac{1}{T_g} \sum_{s=0}^{S-1} \sum_{t=0}^{m-1} \mathbb{E}[\|\nabla f(x_t^{s+1})\|^2] \leq \frac{40Ln^{2/3}\mathbb{E}[f(x^0) - f(x_m^S)]}{T_g}. \tag{4.15}$$

From the definition of (y, z) in output of Algorithm 9 i.e., y is Iterate x_a chosen uniformly random from $\{\{x_t^{s+1}\}_{t=0}^{m-1}\}_{s=0}^{S-1}$ and $z = x_m^S$, it is clear that Algorithm 9 satisfies the **G.2** requirement of GF-OPTIMIZER with $g(n, \epsilon) = T_\epsilon/40Ln^{2/3}$. Since both **G.1** and **G.2** are satisified for Algorithm 9, we conclude that SVRG is a GF-OPTIMIZER. □

4.9 Proof of Lemma 4.4.2

Proof. The first important observation is that the function value never increases because $y = \arg\min_{z \in \{u,x\}} f(z)$ i.e., $f(y) \leq f(x)$, thus satisfying **H.1** of HF-OPTIMIZER. We now analyze the scenario where $\lambda_{min}(\nabla^2 f(x)) \leq -\gamma$. Consider the event where we obtain v such that

$$\langle v, \nabla^2 f(x) v \rangle \leq \lambda_{min}(\nabla^2 f(x)) + \frac{\gamma}{2}.$$

This event (denoted by $\mathcal{E}$) happens with at least probability ρ. Note that, since $\lambda_{min}(\nabla^2 f(x)) \leq -\gamma$, we have $\langle v, \nabla^2 f(x) v \rangle \leq -\frac{\gamma}{2}$. In this case, we have the following relationship:

$$\begin{aligned} f(y) &\leq f(x) + \langle \nabla f(x), y - x \rangle + \frac{1}{2}(y-x)^T \nabla^2 f(x)(y-x) + \frac{M}{6}\|y-x\|^3 \\ &= f(x) - \alpha |\langle \nabla f(x), v \rangle| + \frac{\alpha^2}{2} v^T \nabla^2 f(x) v + \frac{M\alpha^3}{6}\|v\|^3 \\ &\leq f(x) + \frac{\alpha^2}{2} v^T \nabla^2 f(x) v + \frac{M\alpha^3}{6} \\ &\leq f(x) - \frac{1}{2M^2}|v^T \nabla^2 f(x) v|^3 + \frac{1}{6M^2}|v^T \nabla^2 f(x) v|^3 \\ &= f(x) - \frac{1}{3M^2}|v^T \nabla^2 f(x) v|^3 \leq f(x) - \frac{1}{24M^2}\gamma^3. \end{aligned} \tag{4.16}$$

The first inequality follows from the M-lipschitz continuity of the Hessain $\nabla^2 f(x)$. The first equality follows from the update rule of HESSIANDESCENT. The second inequality is obtained by dropping the negative term and using the fact that $\|v\| = 1$. The second equality is obtained by substituting $\alpha = \frac{|v^T \nabla^2 f(x) v|}{M}$. The last inequality is due to the fact that$\langle v, \nabla^2 f(x) v \rangle \leq -\frac{\gamma}{2}$. In the other scenario where

$$\langle v, \nabla^2 f(x) v \rangle \leq \lambda_{min}(\nabla^2 f(x)) + \frac{\gamma}{2},$$

we can at least ensure that $f(y) \leq f(x)$ since $y = \arg\min_{z \in \{u,x\}} f(z)$. Therefore, we have

$$\begin{aligned} \mathbb{E}[f(y)] &= \rho \mathbb{E}[f(y)|\mathcal{E}] + (1-\rho)\mathbb{E}[f(y)|\bar{\mathcal{E}}] \\ &\leq \rho \mathbb{E}[f(y)|\mathcal{E}] + (1-\rho) f(x) \\ &\leq \rho \left[f(x) - \tfrac{\rho}{24M^2}\gamma^3 \right] + (1-\rho) f(x) \\ &= f(x) - \tfrac{\rho}{24M^2}\gamma^3. \end{aligned} \tag{4.17}$$

The last inequality is due to Equation (4.16). Hence, HF-OPTIMIZER satisfies **H.2** of HF-OPTIMIZER with $h(n, \epsilon, \gamma) = \frac{\rho}{24M^2}\gamma^3$, thus concluding the proof. □

4.10 Proof of Theorem 4.4.4

First note that cubic method is a descent method (refer to Theorem 1 of [115]); thus, **H.1** is trivially satisfied. Furthermore, cubic descent is a HF-OPTIMIZER with $h(n, \epsilon, \gamma) = \frac{2\gamma^3}{81M^3}\gamma^3$. This, again, follows from Theorem 1 of [115]. The result easily follows from the aforementioned observations.

4.11 Other Lemmas

The following bound on the variance of SVRG is useful for our proof [139].

Lemma 4.11.1. *[139] Let v_t^{s+1} be computed by Algorithm 9. Then,*

$$\mathbb{E}[\|v_t^{s+1}\|^2] \leq 2\mathbb{E}[\|\nabla f(x_t^{s+1})\|^2] + 2L^2\mathbb{E}[\|x_t^{s+1} - \tilde{x}^s\|^2].$$

Proof. We use the definition of v_t^{s+1} to get

$$\begin{aligned}
&\mathbb{E}[\|v_t^{s+1}\|^2] = \mathbb{E}[\| \left(\nabla f_{i_t}(x_t^{s+1}) - \nabla f_{i_t}(\tilde{x}^s)\right) + \nabla f(\tilde{x}^s)\|^2] \\
&= \mathbb{E}[\| \left(\nabla f_{i_t}(x_t^{s+1}) - \nabla f_{i_t}(\tilde{x}^s)\right) + \nabla f(\tilde{x}^s) - \nabla f(x_t^{s+1}) + \nabla f(x_t^{s+1})\|^2] \\
&\leq 2\mathbb{E}[\|\nabla f(x_t^{s+1})\|^2] + 2\mathbb{E}\left[\left\|\nabla f_{i_t}(x_t^{s+1}) - \nabla f_{i_t}(\tilde{x}^s) - \mathbb{E}[\nabla f_{i_t}(x_t^{s+1}) - \nabla f_{i_t}(\tilde{x}^s)]\right\|^2\right]
\end{aligned}$$

The inequality follows from the simple fact that $(a+b)^2 \leq a^2 + b^2$. From the above inequality, we get the following:

$$\begin{aligned}
\mathbb{E}[\|v_t^{s+1}\|^2] &\leq 2\mathbb{E}[\|\nabla f(x_t^{s+1})\|^2] + 2\mathbb{E}\|\nabla f_{i_t}(x_t^{s+1}) - \nabla f_{i_t}(\tilde{x}^s)\|^2 \\
&\leq 2\mathbb{E}[\|\nabla f(x_t^{s+1})\|^2] + 2L^2\mathbb{E}[\|x_t^{s+1} - \tilde{x}^s\|^2]
\end{aligned}$$

The first inequality follows by noting that for a random variable ζ, $\mathbb{E}[\|\zeta - \mathbb{E}[\zeta]\|^2] \leq \mathbb{E}[\|\zeta\|^2]$. The last inequality follows from L-smoothness of f_{i_t}. □

4.12 Approximate Cubic Regularization

Cubic regularization method of [19] is designed to operate on full batch, i.e., it does not exploit the finite-sum structure of the problem and requires the computation of the gradient and the Hessian on the entire dataset to make an update. However, such full-batch methods do not scale gracefully with the size of data and become prohibitively expensive on large datasets. To overcome this challenge, we devised an approximate cubic regularization method described below:

1. Pick a mini-batch $\mathcal{B}$ and obtain the gradient and the hessian based on $\mathcal{B}$, i.e.,

$$g = \frac{1}{|\mathcal{B}|}\sum_{i\in\mathcal{B}}\nabla f_i(x) \qquad H = \frac{1}{|\mathcal{B}|}\sum_{i\in\mathcal{B}}\nabla^2 f_i(x) \tag{4.18}$$

2. Solve the sub-problem

$$v^* = \arg\min_v \langle g, v\rangle + \frac{1}{2}\langle v, Hv\rangle + \frac{M}{6}\|v\|^3 \tag{4.19}$$

3. Update: $x \leftarrow x + v^*$

We found that this mini-batch training strategy, which requires the computation of the gradient and the Hessian on a small subset of the dataset, to work well on a few datasets (CURVES, MNIST, CIFAR10). A similar method has been analysed in [23].

Furthermore, in many deep-networks, adaptive per-parameter learning rate helps immensely [73]. One possible explanation for this is that the scale of the gradients in each layer of the network often differ by several orders of magnitude. A well-suited optimization method should take this into account. This is the reason for popularity of methods like ADAM or RMSPROP in the deep learning community. On similar lines, to account for different per-parameter behaviour in cubic regularization, we modify the sub-problem by adding a diagonal matrix M_d in addition to the scalar regularization coefficient M, i.e.,

$$\min_v \langle g, v\rangle + \frac{1}{2}\langle v, Hv\rangle + \frac{1}{6}M\|M_d v\|^3. \tag{4.20}$$

Also we devised an adaptive rule to obtain the diagonal matrix as $M_d = \mathrm{diag}((s + 10^{-12})^{1/9})$, where s is maintained as a moving average of third order polynomial of the mini-batch gradient g, in a fashion similar to RMSPROP and ADAM:

$$s \leftarrow \beta s + (1-\beta)(|g|^3 + 2g^2), \tag{4.21}$$

where $|g|^3$ and g^2 are vectors such that $[|g|^3]_i = |g_i|^3$ and $[g^2]_i = g_i^2$ respectively for all $i \in [n]$. The experiments reported on CURVES and MNIST in this chapter utilizes both the above modifications to the cubic regularization, with β set to 0.9. We refer to this modified procedure as ACubic in our results.

4.13 Experiment Details

In this section we provide further experimental details and results to aid reproducibility.

4.13.1 Synthetic Problem

The parameter selection for all the methods were carried as follows:

1. SGD: The scalar step-size was determined by a grid search.

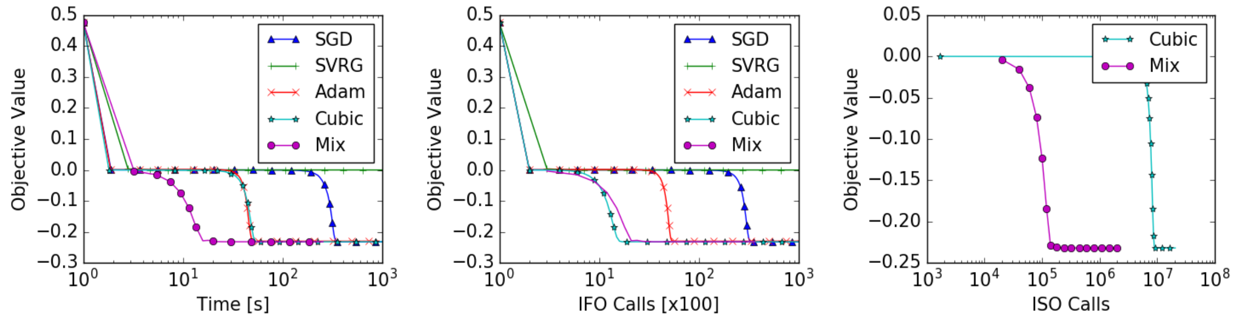


Figure 4.4: Comparison of various methods on a synthetic problem. Our mix framework successfully escapes saddle point.

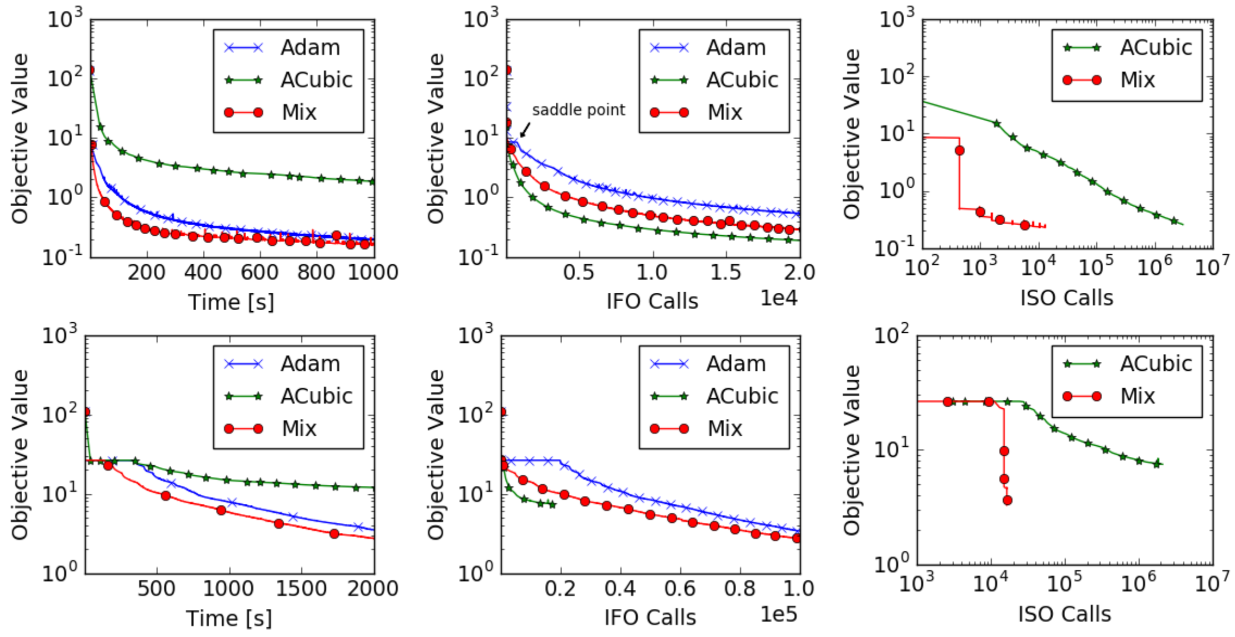


Figure 4.5: Comparison of various methods on a Deep Autoencoder on CURVES (top) and MNIST (bottom). Our mix approach converges faster than the baseline methods and uses relatively few ISO calls in comparison to APPROXCUBICDESCENT

2. ADAM: We performed a grid search over α and ϵ parameters of ADAM tied together, i.e., $\alpha = \epsilon$.
3. SVRG: The scalar step-size was determined by a grid search.
4. CUBICDESCENT: The regularization parameter M was chosen by grid search. The sub-problem was solved with gradient descent [22] with the step-size of solver to be 10^{-2} and run till the gradient norm of the sub-problem is reduced below 10^{-3}.

Further Observations: The results are presented in Figure 4.4. The other first order methods like ADAM with higher noise could escape relatively faster whereas SVRG with reduced noise stayed stuck at the saddle point.

4.13.2 Deep Networks

Methods: The parameter selection for all the methods were carried as follows::

1. ADAM: We performed a grid search over α and ε parameters of ADAM so as to produce the best generalization on a held out test set. We found it to be $\alpha = 10^{-3}, \varepsilon = 10^{-3}$ for CURVES and $\alpha = 10^{-2}, \varepsilon = 10^{-1}$ for MNIST.
2. APPROXCUBICDESCENT: The regularization parameter M was chosen as the largest value such function value does not jump in first 10 epochs. We found it to be $M = 10^3$ for both CURVES and MNIST. The sub-problem was solved with gradient descent [22] with the step-size of solver to be 10^{-3} and run till the gradient norm of the sub-problem is reduced below 0.1.

Chapter 5

Fast Stochastic Methods for Nonsmooth Nonconvex Optimization

5.1 Introduction

In this chapter, we study nonconvex, nonsmooth, finite-sum optimization problems of the form

$$\min_{x\in\mathbb{R}^d} \quad F(x) := f(x) + h(x), \text{ where } f(x) := \frac{1}{n}\sum_{i=1}^{n} f_i(x), \tag{5.1}$$

where each $f_i : \mathbb{R}^d \to \mathbb{R}$ is smooth (possibly nonconvex) for all $i \in \{1, \dots, n\} \triangleq [n]$, while $h : \mathbb{R}^d \to \mathbb{R}$ is nonsmooth but convex and relatively simple.

Such finite-sum optimization problems are fundamental to machine learning, where they typically arise within the spectrum of regularized empirical risk minimization. While there has been extensive research in solving nonsmooth *convex* finite-sum problems (i.e., each f_i is convex for $i \in [n]$) [33, 113, 173], our understanding of their nonsmooth *nonconvex* counterpart is surprisingly limited—despite the widespread use and importance of nonconvex models. We focus, therefore, on fast stochastic methods for solving nonconvex, nonsmooth, finite-sum problems.

A popular approach to handle nonsmoothness is via proximal operators [103, 150]. Recall that for a proper closed convex function h, the *proximal operator* is defined as

$$\mathrm{prox}_{\eta h}(x) := \arg\min_{y\in\mathbb{R}^d} \left(h(y) + \frac{1}{2\eta}\|y - x\|^2 \right), \qquad \text{for } \eta > 0. \tag{5.2}$$

The power of proximal operators lies in how they generalize projections—indeed, if h is the *indicator function* $\mathcal{I}_C(x)$ of a closed convex set C, then $\mathrm{prox}_{\mathcal{I}_C}(x) \equiv \mathrm{proj}_C(x) \equiv \arg\min_{y\in C} \|y - x\|$.

Throughout this chapter, we assume that the proximal operator of h is relatively easy to compute. This is true for many applications in machine learning and statistics including ℓ_1 regularization, box-constraints, simplex constraints, among others [11, 118].

Specifically, we assume access to a *proximal oracle* (PO) that takes a point $x \in \mathbb{R}^d$ and returns the output of (5.2). To describe our complexity results more precisely we use the incremental first-order oracle (IFO).[1] For a function $f = \frac{1}{n}\sum_i f_i$, an IFO takes an index $i \in [n]$ and a point $x \in \mathbb{R}^d$, and returns the pair $(f_i(x), \nabla f_i(x))$.

A standard (batch) method for solving (5.1) is the proximal-gradient method (GD) [102], first studied for nonconvex problems in [43]. This method performs the following iteration:

$$x^{t+1} = \text{prox}_{\eta h}(x^t - \eta \nabla f(x^t)), \qquad t = 0, 1, \ldots, \tag{5.3}$$

where $\eta > 0$ is the step size. The following non-asymptotic rate of convergence result for the proximal gradient method was proved recently.

Theorem (Informal). [48]: *The number of IFO and PO calls made by the proximal gradient method* (5.3) *to reach ϵ close to a stationary point is $O(n/\epsilon)$ and $O(1/\epsilon)$ respectively.*

We refer the readers to [48] for more details. The key point to note here is that the IFO complexity of (5.3) is $O(n/\epsilon)$. This is due to the fact that a full gradient ∇f needs to computed at each iteration of (5.3), thus, entailing n IFO calls at each iteration. When n is large, this per iteration cost is very expensive, and hence often results in slow convergence. A more practical approach is offered by the proximal stochastic gradient (ProxSGD) method, which performs the iteration

$$x^{t+1} = \text{prox}_{\eta_t h}\left(x^t - \frac{\eta_t}{|I_t|}\sum_{i \in I_t} \nabla f_i(x^t)\right), \quad t = 0, 1, \ldots, \tag{5.4}$$

where I_t (referred to as minibatch) is a randomly chosen set (with replacement) from $[n]$ and η_t is a step size. Non-asymptotic convergence of ProxSGD was also shown recently, as noted below.

Theorem (Informal). [48]: *The number of IFO and PO calls made by* ProxSGD, *i.e., iteration* (5.4), *to reach ϵ close to a stationary point is $O(1/\epsilon^2)$ and $O(1/\epsilon)$ respectively. For achieving this convergence, we need batch sizes $|I_t|$ that increase and step sizes η_t that decrease with $1/\epsilon$.*

Notice that the PO complexity of ProxSGD is similar to proximal gradient, but its IFO complexity is independent of n; though this benefit comes at the cost of an extra $1/\epsilon$ factor. Furthermore, the step size must decrease with $1/\epsilon$ (or alternatively decay with the number of iterations of the algorithm). The same two aspects are also seen for *convex* stochastic gradient, in both the smooth and proximal versions. However, in the nonconvex setting there is a key third and more important aspect: *the minibatch size $|I_t|$ increases with $1/\epsilon$.*

To understand this aspect, consider the case of $|I_t|$ being a constant (independent of both n and ϵ), typically the choice used in practice. In this case, the above ProxSGD convergence result no longer holds and it is *not* clear if ProxSGD even converges to a stationary point at all. To clarify, a decreasing step size η_t trivially ensures convergence

[1]Introduced in [2] to study lower bounds of deterministic algorithms for convex finite-sum problems.

as $t \to \infty$, but the limiting point is not necessarily stationary. On the other hand, increasing $|I_t|$ with $1/\epsilon$ can easily lead to $|I_t| \geq n$ for reasonably small ϵ, which effectively reduces the algorithm to (batch) proximal gradient.

This dismal news does not apply to the convex setting, where convergence (in expectation) to an optimal solution has been shown for PROXSGD and its variants using constant minibatch sizes $|I_t|$ [19, 148]. Furthermore, this problem does not afflict the smooth nonconvex case ($h \equiv 0$), where convergence with constant minibatches is ensured [47, 139, 141]. Hence, there appears to be a fundamental gap in our understanding of stochastic methods for *nonsmooth nonconvex* problems. Given the ubiquity of nonconvex models in machine learning and statistics, it is important to bridge this gap. To this end, we study fast stochastic methods for tackling nonsmooth nonconvex problems with guaranteed convergence for constant minibatches, and faster convergence with minibatches independent of $1/\epsilon$.

Main Contributions

We state our main contributions below and list the key complexity results in Table 5.1.

- We analyze nonconvex proximal versions of the recently proposed stochastic algorithms SVRG and SAGA [33, 71, 173], hereafter referred to as PROXSVRG and PROXSAGA, respectively. We show convergence of these algorithms with constant minibatches. To the best of our knowledge, this is the first work to present non-asymptotic convergence rates for stochastic methods that apply to *nonsmooth nonconvex* problems with *constant* (hence more realistic) minibatches.
- We show that by carefully choosing the minibatch size (to be sublinearly dependent on n but still independent of $1/\epsilon$), we can achieve provably faster convergence than both proximal gradient and proximal stochastic gradient. We are not aware of any earlier results on stochastic methods for the general *nonsmooth nonconvex* problem that have faster convergence than proximal gradient.
- We study a nonconvex subclass of (5.1) based on the proximal extension of Polyak-Łojasiewicz inequality [72]. We show linear convergence of PROXSVRG and PROXSAGA to the optimal solution for this subclass. This includes the recent results proved in [154, 179] as special cases. Ours is the first *stochastic* method with provable global linear convergence for this subclass of problems.

5.1.1 Related Work

The literature on finite-sum problems is vast; so we summarize only a few closely related works. Convex instances of (5.1) have been long studied [19, 108, 124] and are fairly well-understood. Remarkable recent progress for smooth convex instances of (5.1) is the creation of variance reduced (VR) stochastic methods [33, 71, 153, 156]. Nonsmooth proximal VR stochastic algorithms are studied in [33, 173] where faster convergence rates for both strongly convex and non-strongly convex cases are proved. Asynchronous VR frameworks are developed in [133]; lower-bounds are studied in [2, 81].

In contrast, nonconvex instances of (5.1) are much less understood. Stochastic gradient for smooth nonconvex problems is analyzed in [47], and only very recently, convergence results for VR stochastic methods for smooth nonconvex problems were obtained in [139, 141]. In [86], the authors consider a VR nonconvex setting different from ours, namely, where the loss is (essentially strongly) convex but hard thresholding is used. We build upon [139, 141], and focus on handling nonsmooth convex regularizers ($h \not\equiv 0$ in (5.1)). Incremental proximal gradient methods for this class were also considered in [164] but only asymptotic convergence was shown. The first analysis of a projection version of nonconvex SVRG is due to [160], who considers the special problem of PCA; see also the follow-up work [161]. Perhaps, the closest to our work is [48], where convergence of minibatch nonconvex PROXSGD method is studied. However, typical to the stochastic gradient method, the convergence is slow; moreover, no convergence for constant minibatches is provided.

5.2 Preliminaries

We assume that the function $h(x)$ in (5.1) is lower semi-continuous (lsc) and convex. Furthermore, we also assume that its domain $\mathrm{dom}(h) = \{x \in \mathbb{R}^d | h(x) < +\infty\}$ is closed. We say f is *L-smooth* if there is a constant L such that

$$\|\nabla f(x) - \nabla f(y)\| \le L\|x - y\|, \quad \forall\, x, y \in \mathbb{R}^d.$$

Throughout, we assume that the functions f_i in (5.1) are L-smooth, so that $\|\nabla f_i(x) - \nabla f_i(y)\| \le L\|x - y\|$ for all $i \in [n]$. Such an assumption is typical in the analysis of first-order methods.

One crucial aspect of the analysis for nonsmooth nonconvex problems is the convergence criterion. For convex problems, typically the optimality gap $F(x) - F(x^*)$ is used as a criterion. It is unreasonable to use such a criterion for general nonconvex problems due to their intractability. For smooth nonconvex problems (i.e., $h \equiv 0$), it is typical to measure stationarity, e.g., using $\|\nabla F\|$. This cannot be used for nonsmooth problems, but a fitting alternative is the *gradient mapping*[2] [111]:

$$\mathcal{G}_\eta(x) := \tfrac{1}{\eta}[x - \mathrm{prox}_{\eta h}(x - \eta \nabla f(x))]. \tag{5.5}$$

When $h \equiv 0$ this mapping reduces to $\mathcal{G}_\eta(x) = \nabla f(x) = \nabla F(x)$, the gradient of function F at x. We analyze our algorithms using the gradient mapping (5.5) as described more precisely below.

Definition 5.2.1. *A point x output by stochastic iterative algorithm for solving* (5.1) *is called an ϵ-accurate solution, if $\mathbb{E}[\|\mathcal{G}_\eta(x)\|^2] \le \epsilon$ for some $\eta > 0$.*

Our goal is to obtain *efficient* algorithms for achieving an ϵ-accurate solution, where efficiency is measured using IFO and PO complexity as functions of $1/\epsilon$ and n.

[2]This mapping has also been used in the analysis of nonconvex proximal methods in [47, 48, 164].

Algorithm	IFO	PO	IFO (PL)	PO (PL)	C-MB?
ProxSgd	$O\left(1/\epsilon^2\right)$	$O\left(1/\epsilon\right)$	$O\left(1/\epsilon^2\right)$	$O\left(1/\epsilon\right)$	?
ProxGD	$O\left(n/\epsilon\right)$	$O\left(1/\epsilon\right)$	$O\left(n\kappa\log(1/\epsilon)\right)$	$O\left(\kappa\log(1/\epsilon)\right)$	–
ProxSvrg	$O(n + (n^{2/3}/\epsilon))$	$O(1/\epsilon)$	$O((n + \kappa n^{2/3})\log(1/\epsilon))$	$O(\kappa\log(1/\epsilon))$	$\surd$
ProxSaga	$O(n + (n^{2/3}/\epsilon))$	$O(1/\epsilon)$	$O((n + \kappa n^{2/3})\log(1/\epsilon))$	$O(\kappa\log(1/\epsilon))$	$\surd$

Table 5.1: Table comparing the *best* IFO and PO complexity of different algorithms discussed in the chapter. The complexity is measured in terms of the number of oracle calls required to achieve an ϵ-accurate solution. The IFO (PL) and PO (PL) represents the IFO and PO complexity of PL functions (see Section 5.5 for a formal definition). The results marked in red highlight our contributions. In the table, "C-MB" indicates whether stochastic algorithm converges using a constant minibatch size. To the best of our knowledge, it is not known if ProxSgd converges on using constant minibatches for nonconvex nonsmooth optimization. Also, we are not aware of any specific convergence results for ProxSgd in the context of PL functions.

5.3 Algorithms

We focus on two algorithms: (a) proximal Svrg (ProxSvrg) and (b) proximal Saga (ProxSaga).

5.3.1 Nonconvex Proximal SVRG

We first consider a variant of ProxSvrg [173]; pseudocode of this variant is stated in Algorithm 11. When F is strongly convex, Svrg attains linear convergence rate as opposed to sublinear convergence of Sgd [71]. Note that, while Svrg is typically stated with $b = 1$, we use its minibatch variant with batch size b. The specific reasons for using such a variant will become clear during the analysis.

While some other algorithms have been proposed for reducing the variance in the stochastic gradients, Svrg is particularly attractive because of its low memory requirement; it requires just $O(d)$ extra memory in comparison to Sgd for storing the average gradient (g^s in Algorithm 11), while algorithms like Sag and Saga incur $O(nd)$ storage cost. In addition to its strong theoretical results, Svrg is known to outperform Sgd empirically while being more robust to selection of step size. For convex problems, ProxSvrg is known to inherit these advantages of Svrg [173].

We now present our analysis of nonconvex ProxSvrg, starting with a result for batch size $b = 1$.

Theorem 5.3.1. *Let $b = 1$ in Algorithm 11. Let $\eta = 1/(3Ln)$, $m = n$ and T be a multiple of m.*

Then the output x_a of Algorithm 11 satisfies the following bound:

$$\mathbb{E}[\|\mathcal{G}_\eta(x_a)\|^2] \leq \frac{18Ln^2}{3n-2}\left(\frac{F(x^0) - F(x^*)}{T}\right),$$

where x^* is an optimal solution of (5.1).

Theorem 5.3.1 shows that PROXSVRG converges for constant minibatches of size $b = 1$. This result is in strong contrast to PROXSGD whose convergence with constant minibatches is still unknown. However, the result delivered by Theorem 5.3.1 is *not* stronger than that of GD. The following corollary to Theorem 5.3.1 highlights this point.

Corollary 5.3.1.1. *To obtain an ϵ-accurate solution, with $b = 1$ and parameters from Theorem 5.3.1, the IFO and PO complexities of Algorithm 5.3.1 are $O(n/\epsilon)$ and $O(n/\epsilon)$, respectively.*

Corollary 5.3.1.1 follows upon noting that each inner iteration (Step 7) of Algorithm 11 has an effective IFO complexity of $O(1)$ since $m = n$. This IFO complexity includes the IFO calls for calculating the average gradient at the end of each epoch. Furthermore, each inner iteration also invokes the proximal oracle, whereby the PO complexity is also $O(n/\epsilon)$. While the IFO complexity of constant minibatch PROXSVRG is same as GD, we see that its PO complexity is much worse. This is due to the fact that n IFO calls correspond to one PO call in GD, while one IFO call in PROXSVRG corresponds to one PO call. Consequently, we do not gain any theoretical advantage by using constant minibatch PROXSVRG over GD.

The key question is therefore: *can we modify the algorithm to obtain better theoretical guarantees?* To answer this question, we prove the following main convergence result. For the ease of theoretical exposition, we assume $n^{2/3}$ to be an integer. This is only for convenience in stating our theoretical results and all the results in the chapter hold for the general case.

Theorem 5.3.2. *Suppose $b = n^{2/3}$ in Algorithm 11. Let $\eta = 1/(3L)$, $m = \lfloor n^{1/3} \rfloor$ and T is a multiple of m. Then for the output x_a of Algorithm 11, we have:*

$$\mathbb{E}[\|\mathcal{G}_\eta(x_a)\|^2] \leq \frac{18L(F(x^0) - F(x^*))}{T},$$

where x^* is an optimal solution to (5.1).

Rewriting Theorem 5.3.2 in terms of the IFO and PO complexity, we obtain the following corollary.

Corollary 5.3.2.1. *Let $b = n^{2/3}$ and set parameters as in Theorem 5.3.2. Then, to obtain an ϵ-accurate solution, the IFO and PO complexities of Algorithm 11 are $O(n + n^{2/3}/\epsilon)$ and $O(1/\epsilon)$, respectively.*

The above corollary is due to the following observations. From Theorem 5.3.2, it can be seen that the total number of inner iterations (across all epochs) of Algorithm 11 to obtain an ϵ-accurate solution is $O(1/\epsilon)$. Since each inner iteration of Algorithm 5.3.2 involves a call to the PO, we obtain a PO complexity of $O(1/\epsilon)$. Further, since $b = n^{2/3}$

Algorithm 11: Nonconvex ProxSvrg (x^0, T, m, b, η)

```
1:  Input: $\tilde{x}^0 = x_m^0 = x^0 \in \mathbb{R}^d$, epoch length $m$, step sizes $\eta > 0$, $S = \lceil T/m \rceil$
2:  for $s = 0$ to $S-1$ do
3:      $x_0^{s+1} = x_m^s$
4:      $g^{s+1} = \frac{1}{n}\sum_{i=1}^n \nabla f_i(\tilde{x}^s)$
5:      for $t = 0$ to $m-1$ do
6:          Uniformly randomly pick $I_t \subset \{1, \dots, n\}$ (with replacement) such that $|I_t| = b$
7:          $v_t^{s+1} = \frac{1}{b}\sum_{i_t \in I_t}(\nabla f_{i_t}(x_t^{s+1}) - \nabla f_{i_t}(\tilde{x}^s)) + g^{s+1}$
8:          $x_{t+1}^{s+1} = \text{prox}_{\eta h}(x_t^{s+1} - \eta v_t^{s+1})$
9:      end for
10:     $\tilde{x}^{s+1} = x_m^{s+1}$
11: end for
12: Output: Iterate $x_a$ chosen uniformly at random from $\{\{x_t^{s+1}\}_{t=0}^{m-1}\}_{s=0}^{S-1}$.
```

IFO calls are made at each inner iteration, we obtain a net IFO complexity of $O(n^{2/3}/\epsilon)$. Adding the IFO calls for the calculation of the average gradient (and noting that T is a multiple of m), we obtain the desired result. A noteworthy aspect of Corollary 5.3.2.1 is that its PO complexity matches GD, but its IFO complexity is significantly decreased to $O(n + n^{2/3}/\epsilon)$ as opposed to $O(n/\epsilon)$ in GD.

5.3.2 Nonconvex Proximal SAGA

In the previous section, we investigated ProxSvrg for solving (5.1). Note that ProxSvrg is not a fully "incremental" algorithm since it requires calculation of the full gradient once per epoch. An alternative to ProxSvrg is the algorithm proposed in [33] (popularly referred to as SAGA). We build upon the work of [33] to develop ProxSaga, a nonconvex proximal variant of Saga.

The pseudocode for ProxSaga is presented in Algorithm 12. The key difference between Algorithm 11 and 12 is that ProxSaga, unlike ProxSvrg, avoids computation of the full gradient. Instead, it maintains an average gradient vector g^t, which changes at each iteration (refer to [133]). However, such a strategy entails additional storage costs. In particular, for implementing Algorithm 12, we must store the gradients $\{\nabla f_i(\alpha_i^t)\}_{i=1}^n$, which in general can cost $O(nd)$ in storage. Nevertheless, in some scenarios common to machine learning (see [33]), one can reduce the storage requirements to $O(n)$. Whenever such an implementation of ProxSaga is possible, it can perform similar to or even better than ProxSvrg [33]; hence, in addition to theoretical interest, it is of significant practical value.

We remark that ProxSaga in Algorithm 12 differs slightly from [33]. In particular, it uses minibatches where two sets I_t, J_t are sampled at each iteration as opposed to one in [33]. This is mainly for the ease of theoretical analysis.

We prove that as in the convex case, nonconvex ProxSvrg and ProxSaga share similar theoretical guarantees. In particular, our first result for ProxSaga is a counterpart to Theorem 5.3.1 for ProxSvrg.

Algorithm 12: Nonconvex ProxSaga (x^0, T, b, η)

```
1: Input: $x^0 \in \mathbb{R}^d$, $\alpha_i^0 = x^0$ for $i \in [n]$, step size $\eta > 0$
2: $g^0 = \frac{1}{n}\sum_{i=1}^n \nabla f_i(\alpha_i^0)$
3: for $t = 0$ to $T-1$ do
4:     Uniformly randomly pick sets $I_t, J_t$ from $[n]$ (with replacement) such that $|I_t| = |J_t| = b$
5:     $v^t = \frac{1}{b}\sum_{i_t \in I_t}(\nabla f_{i_t}(x^t) - \nabla f_{i_t}(\alpha_{i_t}^t)) + g^t$
6:     $x^{t+1} = \text{prox}_{\eta h}(x^t - \eta v^t)$
7:     $\alpha_j^{t+1} = x^t$ for $j \in J_t$ and $\alpha_j^{t+1} = \alpha_j^t$ for $j \notin J_t$
8:     $g^{t+1} = g^t - \frac{1}{n}\sum_{j_t \in J_t}(\nabla f_{j_t}(\alpha_{j_t}^t) - \nabla f_{j_t}(\alpha_{j_t}^{t+1}))$
9: end for
10: Output: Iterate $x_a$ chosen uniformly random from $\{x^t\}_{t=0}^{T-1}$.
```

Theorem 5.3.3. *Suppose $b = 1$ in Algorithm 12. Let $\eta = 1/(5Ln)$. Then for the output x_a of Algorithm 12 after T iterations, the following stationarity bound holds:*

$$\mathbb{E}[\|\mathcal{G}_\eta(x_a)\|^2] \le \frac{50Ln^2}{5n-2}\frac{F(x^0) - F(x^*)}{T},$$

where x^* is an optimal solution of (5.1).

Theorem 5.3.3 immediately leads to the following corollary.

Corollary 5.3.3.1. *The IFO and PO complexity of Algorithm 5.3.3 for $b = 1$ and parameters specified in Theorem 5.3.3 to obtain an ϵ-accurate solution are $O(n/\epsilon)$ and $O(n/\epsilon)$ respectively.*

Similar to Theorem 5.3.2 for ProxSvrg, we obtain the following main result for ProxSaga.

Theorem 5.3.4. *Suppose $b = n^{2/3}$ in Algorithm 12. Let $\eta = 1/(5L)$. Then for the output x_a of Algorithm 12 after T iterations, the following holds:*

$$\mathbb{E}[\|\mathcal{G}_\eta(x_a)\|^2] \le \frac{50L(F(x^0) - F(x^*))}{3T},$$

where x^* is an optimal solution of Problem (5.1).

Rewriting this result in terms of IFO and PO access, we obtain the following important corollary.

Corollary 5.3.4.1. *Let $b = n^{2/3}$ and set parameters as in Theorem 5.3.4. Then, to obtain an ϵ-accurate solution, the IFO and PO complexities of Algorithm 12 are $O(n + n^{2/3}/\epsilon)$ and $O(1/\epsilon)$, respectively.*

The above result is due to Theorem 5.3.4 and because each iteration of ProxSaga requires $O(n^{2/3})$ IFO calls. The number of PO calls is only $O(1/\epsilon)$, since make one PO call for every $n^{2/3}$ IFO calls.

Discussion: It is important to note the role of minibatches in Corollaries 5.3.2.1 and 5.3.4.1. Minibatches are typically used for reducing variance and promoting parallelism in stochastic methods. But unlike previous works, we use minibatches as a theoretical tool to improve convergence rates of both nonconvex ProxSvrg and ProxSaga. In particular, by carefully selecting the minibatch size, we can improve the IFO complexity of the algorithms described in the chapter from $O(n/\epsilon)$ (similar to GD) to $O(n^{2/3}/\epsilon)$ (matching our results for the smooth nonconvex case in [139, 141]). Furthermore, the PO complexity is also improved in a similar manner by using the minibatch size mentioned in Theorems 5.3.2 and 5.3.4.

5.4 General Convergence Analysis

In this previous sections, for the sake of clarity, we stated convergence rates of ProxSvrg and ProxSaga for a specific set of parameters. However, a more general analysis can be derived for these algorithms. The rationale behind the choice of parameters is Section 5.3 will also become clear later in this section. We have the following general convergence results for ProxSvrg and ProxSaga.

Theorem 5.4.1. *Suppose $b \leq n$ in Algorithm 11. Let T be a multiple of m and $\eta = \rho/L$ where $\rho < 1/2$ and satisfies the following:*

$$\frac{4\rho^2 m^2}{b} + \rho \leq 1.$$

Then for the output x_a of Algorithm 11, we have:

$$\mathbb{E}[\|\mathcal{G}_\eta(x_a)\|^2] \leq \frac{2L(F(x^0) - F(x^*))}{\rho(1-2\rho)T},$$

where x^* is an optimal solution to (5.1).

The following result is an immediate consequence of the above result.

Corollary 5.4.1.1. *Let $b \leq n$, $\rho = 1/4$ and $m = \lfloor b^{1/2} \rfloor$ in Theorem 5.4.1. Then, to obtain an ϵ-accurate solution, the IFO and PO complexities of Algorithm 11 are $O(n + n/(b^{1/2}\epsilon) + b/\epsilon)$ and $O(1/\epsilon)$, respectively.*

We observe that under the parameter setting in Corollary 5.4.1.1, the PO complexity is $O(1/\epsilon)$, which matches that of GD. Thus, this setting is optimized for reducing the PO complexity. Furthermore, for constant minibatch $b = 1$, Corollary 5.4.1.1 shows that the IFO and PO complexity of ProxSvrg is $O(n/\epsilon)$ and $O(1/\epsilon)$ respectively, which is stronger than Theorem 5.3.1. However, in the setting of Corollary 5.4.1.1 with minibatch size $b = 1$, ProxSvrg effectively reduces to GD since $m = \lfloor b^{1/2} \rfloor = 1$ and hence, not very interesting. When $b = 1$, Theorem 5.3.1 provides the convergence result of ProxSvrg for the setting that is generally used in practice where $m = n$.

For ProxSaga, we have the following general convergence results.

Theorem 5.4.2. *Suppose $b \leq n$ in Algorithm 12. Let $\eta = \rho/L$ where $\rho < 1/2$ and ρ satisfies the following condition:*

$$\frac{16n^2\rho^2}{b^3} + \rho \leq 1.$$

```
PL-SVRG:$(x^0, K, T, m, \eta)$
for $k = 1$ to $K$ do
    $x^k = \text{ProxSVRG}(x^{k-1}, T, m, b, \eta)$ ;
end
Output: $x^K$
```

```
PL-SAGA:$(x^0, K, T, m, \eta)$
for $k = 1$ to $K$ do
    $x^k = \text{ProxSAGA}(x^{k-1}, T, b, \eta)$ ;
end
Output: $x^K$
```

Figure 5.1: PROXSVRG and PROXSAGA variants for PL functions.

Then for the output x_a of Algorithm 12, we have:

$$\mathbb{E}[\|\mathcal{G}_\eta(x_a)\|^2] \leq \frac{2L(F(x^0) - F(x^*))}{\rho(1 - 2\rho)T},$$

where x^* is an optimal solution to (5.1).

Corollary 5.4.2.1. *Let $b \leq n$, $\rho = \min\{1/5, b^{3/2}/5n\}$ in Theorem 5.4.2. Then, to obtain an ϵ-accurate solution, the IFO and PO complexities of Algorithm 12 are $O(n + n/(b^{1/2}\epsilon) + b/\epsilon)$ and $O(\max\{1, n/b^{3/2}\}/\epsilon)$ respectively.*

Note that the IFO complexity of PROXSVRG (in Corollary 5.4.1.1) and PROXSAGA (in Corollary 5.4.2.1) are similar; however, their PO complexities are different. It is not hard to see from Corollary 5.4.1.1 and 5.4.2.1, that the best IFO and PO complexity of both PROXSVRG and PROXSAGA obtainable through these upper bounds are $O(n + n^{2/3}/\epsilon)$ and $O(1/\epsilon)$ respectively; which are precisely our main results in Section 5.3.

5.5 Extensions

We discuss some extensions of our approach in this section. Our first extension is to provide convergence analysis for a subclass of nonconvex functions that satisfy a specific growth condition popularly known as the Polyak-Łojasiewicz (PL) inequality. In the context of gradient descent, this inequality was proposed by Polyak in 1963 [122], who showed *global* linear convergence of gradient descent for functions that satisfy the PL inequality. Recently, in [72] the PL inequality was generalized to nonsmooth functions and used for proving linear convergence of proximal gradient. The generalization presented in [72] considers functions $F(x) = f(x) + h(x)$ that satisfy the following:

$$\begin{aligned} &\mu(F(x) - F(x^*)) \leq \frac{1}{2}D_h(x, \mu), \text{ where } \mu > 0 \\ &\text{and } D_h(x, \mu) := -2\mu \min_y \left[\langle \nabla f(x), y - x\rangle + \frac{\mu}{2}\|y - x\|^2 + h(y) - h(x)\right]. \end{aligned} \tag{5.6}$$

An F that satisfies (5.6) is called a μ-PL function.

When $h \equiv 0$, condition (5.6) reduces to the usual PL inequality. The class of μ-PL functions includes several other classes as special cases. It subsumes strongly convex functions, covers $f_i(x) = g(a_i^\top x)$ with only g being strongly convex, and includes functions that satisfy a optimal strong convexity property [89]. Note that the μ-PL functions also subsume the recently studied special case where f_i's are nonconvex but their sum f is strongly convex. Hence, it encapsulates the problems of [154, 179].

The algorithms in Figure 5.1 provide variants of ProxSvrg and ProxSaga adapted to optimize μ-PL functions. We show the following *global* linear convergence result of Pl-Svrg and Pl-Saga in Figure 5.1 for PL functions. For simplicity, we assume $\kappa = (L/\mu) > n^{1/3}$. When f is strongly convex, κ is referred to as the condition number, in which case $\kappa > n^{1/3}$ corresponds to the high condition number regime.

Theorem 5.5.1. *Suppose F is a μ-PL function. Let $b = n^{2/3}$, $\eta = 1/5L$, $m = \lfloor n^{1/3} \rfloor$ and $T = \lceil 30\kappa \rceil$. Then for the output x^K of* Pl-Svrg *and* Pl-Saga *(in Figure 5.1), the following holds:*

$$\mathbb{E}[F(x^K) - F(x^*)] \leq \frac{[F(x^0) - F(x^*)]}{2^K},$$

where x^* is an optimal solution of (5.1).

The following corollary on IFO and PO complexity of Pl-Svrg and Pl-Saga is immediate.

Corollary 5.5.1.1. *When F is a μ-PL function, then the IFO and PO complexities of* Pl-Svrg *and* Pl-Saga *with the parameters specified in Theorem 5.5.1 to obtain an ϵ-accurate solution are $O((n + \kappa n^{2/3}) \log(1/\epsilon))$ and $O(\kappa \log(1/\epsilon))$, respectively.*

Note that proximal gradient also has global linear convergence for PL functions, as recently shown in [72]. However, its IFO complexity is $O(\kappa n \log(1/\epsilon))$, which is much worser than that of Pl-Svrg and Pl-Saga (Corollary 5.5.1.1).

Other extensions: Our results can be easily generalized to the case where non-uniform sampling is used in Algorithm 11 and Algorithm 12. This is useful when the functions f_i have different Lipschitz constants.

5.6 Experiments

We present our empirical results in this section. For our experiments, we study the problem of non-negative principal component analysis (NN-PCA). More specifically, for a given set of samples $\{z_i\}_{i=1}^n$, we solve the following optimization problem:

$$\min_{\|x\| \leq 1,\, x \geq 0} -\frac{1}{2} x^\top \left(\sum_{i=1}^n z_i z_i^\top \right) x. \tag{5.7}$$

The problem of NN-PCA is, in general, NP-hard. This variant of the standard PCA problem can be written in the form (5.1) with $f_i(x) = -(x^\top z_i)^2$ for all $i \in [n]$ and $h(x) = \mathcal{I}_C(x)$ where C is the convex set $\{x \in \mathbb{R}^d \mid \|x\| \leq 1, x \geq 0\}$. In our experiments, we compare ProxSgd with nonconvex ProxSvrg and ProxSaga. The choice of step

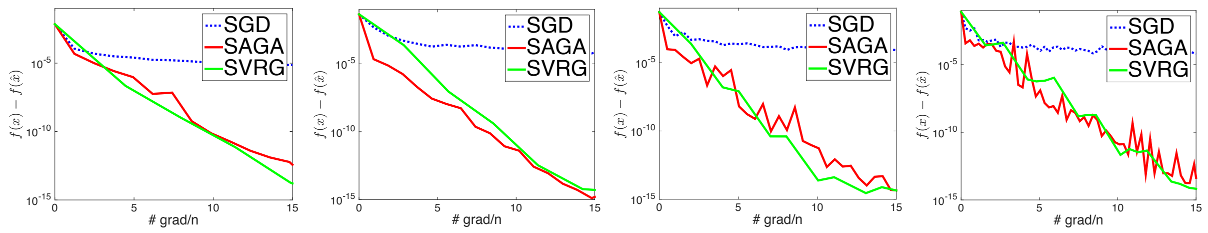


Figure 5.2: Non-negative principal component analysis. Performance of PROXSGD, PROXSVRG and PROXSAGA on 'rcv1' (left), 'a9a'(left-center), 'mnist' (right-center) and 'aloi' (right) datasets. Here, the y-axis is the function suboptimality i.e., $f(x) - f(\hat{x})$ where $\hat{x}$ represents the best solution obtained by running gradient descent for long time and with multiple restarts.

size is important to PROXSGD. The step size of PROXSGD is set using the popular t-inverse step size choice of $\eta_t = \eta_0(1 + \eta' \lfloor t/n \rfloor)^{-1}$ where η_0, $\eta' > 0$. For PROXSVRG and PROXSAGA, motivated by the theoretical analysis, we use a fixed step size. The parameters of the step size in each of these methods are chosen so that the method gives the best performance on the objective value. In our experiments, we include the value $\eta' = 0$, which corresponds to PROXSGD with fixed step size. For PROXSVRG, we use the epoch length $m = n$.

We use standard machine learning datasets in LIBSVM for all our experiments [3]. The samples from each of these datasets are normalized i.e. $\|z_i\| = 1$ for all $i \in [n]$. Each of these methods is initialized by running PROXSGD for n iterations. Such an initialization serves two purposes: (a) it provides a reasonably good initial point, typically beneficial for variance reduction techniques [33, 153]. (b) it provides a heuristic for calculating the initial average gradient g^0 [153]. In our experiments, we use minibatch size $b = 1$ in order to demonstrate the performance of the algorithms with constant minibatches.

We report the objective function value for the datasets. In particular, we report the suboptimality in objective function i.e., $f(x_t^{s+1}) - f(\hat{x})$ (for PROXSVRG) and $f(x^t) - f(\hat{x})$ (for PROXSAGA). Here $\hat{x}$ refers to the solution obtained by running proximal gradient descent for a large number of iterations and multiple random initializations. For all the algorithms, we compare the aforementioned criteria against for the number of *effective* passes through the dataset i.e., IFO complexity divided by n. For PROXSVRG, this includes the cost of calculating the full gradient at the end of each epoch.

Figure 5.2 shows the performance of PROXSGD , PROXSVRG and PROXSVRG on NN-PCA problem (see Section 5.11 of the Appendix for more experiments). It can be seen that the objective value for PROXSVRG and PROXSAGA is much lower compared to PROXSGD, suggesting faster convergence for these algorithms. We observed a significant gain consistently across all the datasets. Moreover, the selection of step size was much simpler for PROXSVRG and PROXSAGA than that for PROXSGD. We did not observe any significant difference in the performance of PROXSVRG and PROXSAGA for

[3]The datasets can be downloaded from https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets.

this particular task.

5.7 Discussion

In this chapter, we presented fast stochastic methods for nonsmooth nonconvex optimization. In particular, by employing variance reduction techniques, we show that one can design methods that can provably perform better than ProxSgd and proximal gradient descent. Furthermore, in contrast to ProxSgd, the resulting approaches have provable convergence to a stationary point with constant minibatches; thus, bridging a fundamental gap in our knowledge of nonsmooth nonconvex problems.

We proved that with a careful selection of minibatch size, it is possible to theoretically show superior performance to proximal gradient descent. Our empirical results provide evidence for a similar conclusion even with constant minibatches. Thus, we conclude with an important open problem of developing stochastic methods with provably better performance than proximal gradient descent with *constant minibatch* size.

Appendix: Omitted Proofs and Additional Experiments

5.8 Convergence analysis for Proximal Nonconvex Svrg

The analysis requires some key lemmas which can be found in Appendix 5.12.

5.8.1 General Convergence Analysis: Proof of Theorem 5.4.1

Proof. We start by defining the full gradient iterate

$$\bar{x}_{t+1}^{s+1} = \mathrm{prox}_{\eta h}(x_t^{s+1} - \eta \nabla f(x_t^{s+1})), \tag{5.8}$$

which is merely for our analysis, and is never actually computed. Applying Lemma 5.12.2 to (5.8) (with $y = \bar{x}_{t+1}^{s+1}$, $z = x_t^{s+1}$ and $d' = \nabla f(x_t^{s+1})$), and taking expectations we obtain the bound

$$\mathbb{E}[F(\bar{x}_{t+1}^{s+1})] \le \mathbb{E}\left[F(x_t^{s+1}) + \left[\frac{L}{2} - \frac{1}{2\eta}\right] \|\bar{x}_{t+1}^{s+1} - x_t^{s+1}\|^2 - \frac{1}{2\eta}\|\bar{x}_{t+1}^{s+1} - x_t^{s+1}\|^2\right]. \tag{5.9}$$

Recall the iterates of Algorithm 11 are computed using the following update:

$$x_{t+1}^{s+1} = \mathrm{prox}_{\eta h}(x_t^{s+1} - \eta v_t^{s+1})), \tag{5.10}$$

where $v_t^{s+1} = \frac{1}{b}\sum_{i_t \in I_t}(\nabla f_{i_t}(x_t^{s+1}) - \nabla f_{i_t}(\tilde{x}^s)) + g^{s+1}$ (see Algorithm 11). Applying Lemma 5.12.2 on update (5.10) (with $y = x_{t+1}^{s+1}$, $z = \bar{x}_{t+1}^{s+1}$ and $d' = v_t^{s+1}$) and taking expectations we obtain

$$\mathbb{E}[F(x_{t+1}^{s+1})] \leq \mathbb{E}\Big[F(\overline{x}_{t+1}^{s+1}) + \langle x_{t+1}^{s+1} - \overline{x}_{t+1}^{s+1}, \nabla f(x_t^{s+1}) - v_t^{s+1}\rangle$$
$$+ \left[\tfrac{L}{2} - \tfrac{1}{2\eta}\right] \|x_{t+1}^{s+1} - x_t^{s+1}\|^2 + \left[\tfrac{L}{2} + \tfrac{1}{2\eta}\right] \|\overline{x}_{t+1}^{s+1} - x_t^{s+1}\|^2 - \tfrac{1}{2\eta}\|x_{t+1}^{s+1} - \overline{x}_{t+1}^{s+1}\|^2\Big]. \tag{5.11}$$

Adding inequalities (5.9) and (5.11), we get

$$\mathbb{E}[F(x_{t+1}^{s+1})] \leq \mathbb{E}\Big[F(x_t^{s+1}) + \left[L - \tfrac{1}{2\eta}\right] \|\overline{x}_{t+1}^{s+1} - x_t^{s+1}\|^2 + \left[\tfrac{L}{2} - \tfrac{1}{2\eta}\right] \|x_{t+1}^{s+1} - x_t^{s+1}\|^2$$
$$- \tfrac{1}{2\eta}\|x_{t+1}^{s+1} - \overline{x}_{t+1}^{s+1}\|^2 + \underbrace{\langle x_{t+1}^{s+1} - \overline{x}_{t+1}^{s+1}, \nabla f(x_t^{s+1}) - v_t^{s+1}\rangle}_{T_1}\Big] \tag{5.12}$$

We can bound the term T_1 as follows:

$$\mathbb{E}[T_1] \leq \frac{1}{2\eta}\mathbb{E}\|x_{t+1}^{s+1} - \overline{x}_{t+1}^{s+1}\|^2 + \frac{\eta}{2}\mathbb{E}\|\nabla f(x_t^{s+1}) - v_t^{s+1}\|^2$$
$$\leq \frac{1}{2\eta}\mathbb{E}\|x_{t+1}^{s+1} - \overline{x}_{t+1}^{s+1}\|^2 + \frac{\eta L^2}{2b}\mathbb{E}\|x_t^{s+1} - \tilde{x}^s\|^2.$$

The first inequality follows from Cauchy-Schwarz and Young's inequality, while the second inequality is due to Lemma 5.12.3. Substituting the upper bound on T_1 in (5.12), we see that

$$\mathbb{E}[F(x_{t+1}^{s+1})] \leq \mathbb{E}\Big[F(x_t^{s+1}) + \left[L - \tfrac{1}{2\eta}\right] \|\overline{x}_{t+1}^{s+1} - x_t^{s+1}\|^2$$
$$+ \left[\tfrac{L}{2} - \tfrac{1}{2\eta}\right] \|x_{t+1}^{s+1} - x_t^{s+1}\|^2 + \tfrac{\eta L^2}{2b}\|x_t^{s+1} - \tilde{x}^s\|^2\Big]. \tag{5.13}$$

To further analyze (5.13), we set up a recursion for which we use the following Lyapunov function:

$$R_t^{s+1} := \mathbb{E}[F(x_t^{s+1}) + c_t\|x_t^{s+1} - \tilde{x}^s\|^2].$$

Introduce the quantities $c_m = 0$, and $c_t = c_{t+1}(1+\beta) + \frac{\eta L^2}{2b}$. Also, for rest of the analysis set $\beta = 1/m$. We can then bound R_{t+1}^{s+1} as follows

$$R_{t+1}^{s+1} = \mathbb{E}[F(x_{t+1}^{s+1}) + c_{t+1}\|x_{t+1}^{s+1} - x_t^{s+1} + x_t^{s+1} - \tilde{x}^s\|^2]$$
$$= \mathbb{E}[F(x_{t+1}^{s+1}) + c_{t+1}(\|x_{t+1}^{s+1} - x_t^{s+1}\|^2 + \|x_t^{s+1} - \tilde{x}^s\|^2 + 2\langle x_{t+1}^{s+1} - x_t^{s+1}, x_t^{s+1} - \tilde{x}^s\rangle)]$$
$$\leq \mathbb{E}[F(x_{t+1}^{s+1}) + c_{t+1}(1 + 1/\beta)\|x_{t+1}^{s+1} - x_t^{s+1}\|^2 + c_{t+1}(1+\beta)\|x_t^{s+1} - \tilde{x}^s\|^2]$$
$$\leq \mathbb{E}\Big[F(x_t^{s+1}) + \left[L - \tfrac{1}{2\eta}\right] \|\overline{x}_{t+1}^{s+1} - x_t^{s+1}\|^2 + \left[c_{t+1}\left(1 + \tfrac{1}{\beta}\right) + \tfrac{L}{2} - \tfrac{1}{2\eta}\right] \|x_{t+1}^{s+1} - x_t^{s+1}\|^2$$
$$+ \left[c_{t+1}(1+\beta) + \tfrac{\eta L^2}{2b}\right] \|x_t^{s+1} - \tilde{x}^s\|^2\Big] \tag{5.14}$$
$$\leq \mathbb{E}\Big[F(x_t^{s+1}) + \left[L - \tfrac{1}{2\eta}\right] \|\overline{x}_{t+1}^{s+1} - x_t^{s+1}\|^2 + \left[c_{t+1}(1+\beta) + \tfrac{\eta L^2}{2b}\right] \|x_t^{s+1} - \tilde{x}^s\|^2\Big]$$
$$= R_t^{s+1} + \left[L - \tfrac{1}{2\eta}\right] \mathbb{E}\|\overline{x}_{t+1}^{s+1} - x_t^{s+1}\|^2. \tag{5.15}$$

The first inequality follows from Cauchy-Schwarz and Young's inequality. The second inequality is due to the bound (5.13), while the final equality is due to the definition of the Lyapunov function R_t^{s+1}. The third inequality holds because the sequence of values c_t satisfies the following bound:

$$c_{t+1}\left(1+\frac{1}{\beta}\right)+\frac{L}{2}\leq\frac{1}{2\eta}. \tag{5.16}$$

To verify (5.16), first observe that $c_m = 0$ and $c_t = c_{t+1}(1+\beta)+\frac{\eta L^2}{2b}$. Recursing on t, we thus obtain

$$\begin{aligned} c_t = \frac{\eta L^2}{2b}\frac{(1+\beta)^{m-t}-1}{\beta} &= \frac{\rho L m}{2b}\left(\left(1+\frac{1}{m}\right)^{m-t}-1\right) \\ &\leq \frac{\rho L m}{2b}(e-1)\leq\frac{\rho L m}{b}, \end{aligned}$$

where the first equality is due to the definition of η and β. The first inequality follows upon noting that (i) $\lim_{l\to+\infty}(1+1/l)^l = e$ and (ii) $(1+1/l)^l$ is an increasing function for $l > 0$ (here e is Euler's number). It follows that

$$\begin{aligned} c_{t+1}\left(1+\frac{1}{\beta}\right)+\frac{L}{2} &\leq \frac{\rho L m}{b}(1+m)+\frac{L}{2} \\ &\leq \frac{2\rho L m^2}{b}+\frac{L}{2}\leq\frac{L}{2\rho}=\frac{1}{2\eta}, \end{aligned}$$

where the second inequality uses $m \geq 1$. The third inequality uses the condition that

$$\frac{4\rho^2 m^2}{b}+\rho\leq 1.$$

Hence, inequality (5.16) follows. Now, adding (5.15) across all the iterations in epoch $s+1$ and then telescoping sums, we get

$$R_m^{s+1}\leq R_0^{s+1}+\textstyle\sum_{t=0}^{m-1}\left[L-\frac{1}{2\eta}\right]\mathbb{E}\|\overline{x}_{t+1}^{s+1}-x_t^{s+1}\|^2. \tag{5.17}$$

Since $c_m = 0$ and from the definition of $\tilde{x}^{s+1}$, it follows that $R_m^{s+1} = \mathbb{E}[F(x_m^{s+1})] = \mathbb{E}[F(\tilde{x}^{s+1})]$. Furthermore, $R_0^{s+1} = \mathbb{E}[F(x_0^{s+1})] = \mathbb{E}[F(\tilde{x}^s)]$. This is due to the fact that $x_0^{s+1} = \tilde{x}^s$. Therefore, using the above reasoning in inequality (5.17), we have

$$\mathbb{E}[F(\tilde{x}^{s+1})]\leq\mathbb{E}[F(\tilde{x}^s)]+\sum_{t=0}^{m-1}\left[L-\tfrac{1}{2\eta}\right]\mathbb{E}\|\overline{x}_{t+1}^{s+1}-x_t^{s+1}\|^2. \tag{5.18}$$

Adding (5.18) across all the epochs and rearranging the terms, we obtain the bound

$$\sum_{s=0}^{S}\sum_{t=0}^{m-1}\left[\tfrac{1}{2\eta}-L\right]\mathbb{E}\|\overline{x}_{t+1}^{s+1}-x_t^{s+1}\|^2\leq F(x^0)-\mathbb{E}[F(\tilde{x}^S)]\leq F(x^0)-F(x^*), \tag{5.19}$$

where the second inequality follows from the optimality of x^*.

Recall that in our notation

$$\mathcal{G}_\eta(x_t^{s+1}) = \frac{1}{\eta}[x_t^{s+1} - \text{prox}_{\eta h}(x_t^{s+1} - \eta \nabla f(x_t^{s+1}))] = \frac{1}{\eta}[x_t^{s+1} - \overline{x}_{t+1}^{s+1}].$$

Using this relationship in (5.19) we see that

$$\sum_{s=0}^{S} \sum_{t=0}^{m-1} \left[\frac{1}{2\eta} - L\right] \eta^2 \mathbb{E}\|\mathcal{G}_\eta(x_t^{s+1})\|^2 \le F(x^0) - F(x^*). \tag{5.20}$$

Now using the definition of x_a from Algorithm 11 and simplifying we obtain the desired result. □

Proof of Theorem 5.3.1

Proof. The proof follows from Theorem 5.4.1 with $b = 1$ and the parameters used in the theorem statement. □

Proof of Theorem 5.3.2

Proof. The proof follows from Theorem 5.4.1 with $b = n^{2/3}$ and the parameters used in the theorem statement. □

5.9 Convergence analysis for Nonconvex Proximal SAGA

5.9.1 General Convergence Analysis: Proof of Theorem 5.4.2

Proof. We introduce the full-gradient iterate (as before, only for the analysis)

$$\overline{x}^{t+1} = \text{prox}_{\eta h}(x^t - \eta \nabla f(x^t)), \tag{5.21}$$

and recall that PROXSAGA iterations compute the update

$$x^{t+1} = \text{prox}_{\eta h}(x^t - \eta v^t),$$

where $v^t = \frac{1}{b}\sum_{i_t \in I_t}\left(\nabla f_{i_t}(x^t) - \nabla f_{i_t}(\alpha_{i_t}^t)\right) + g^t$. Now, using the same argument as in Theorem 5.4.1 until inequality (5.12), we obtain the following

$$\begin{aligned}\mathbb{E}[F(x^{t+1})] \le \mathbb{E}\Big[F(x^t) &+ \left[L - \tfrac{1}{2\eta}\right] \|\overline{x}^{t+1} - x^t\|^2 + \left[\tfrac{L}{2} - \tfrac{1}{2\eta}\right] \|x^{t+1} - x^t\|^2 \\ &- \tfrac{1}{2\eta}\|x^{t+1} - \overline{x}^{t+1}\|^2 + \underbrace{\langle x^{t+1} - \overline{x}^{t+1}, \nabla f(x^t) - v^t\rangle}_{T_2}\Big].\end{aligned} \tag{5.22}$$

The term T_2 in (5.22) can be bound as follows:

$$\mathbb{E}[T_2] \leq \frac{1}{2\eta}\mathbb{E}\|x^{t+1} - \overline{x}^{t+1}\|^2 + \frac{\eta}{2}\mathbb{E}\|\nabla f(x^t) - v^t\|^2 \leq \frac{1}{2\eta}\mathbb{E}\|x^{t+1} - \overline{x}^{t+1}\|^2 + \frac{\eta L^2}{2nb}\sum_{i=1}^{n}\mathbb{E}\|x^t - \alpha_i^t\|^2.$$

The inequality follows from Cauchy-Schwarz and Young's inequality. The second inequality is due to Lemma 5.12.4. Substituting the upper bound on T_2 in inequality (5.22), we have

$$\begin{aligned}\mathbb{E}[F(x^{t+1})] \leq \mathbb{E}\Big[F(x^t) + \Big[L - \tfrac{1}{2\eta}\Big]\|\overline{x}^{t+1} - x^t\|^2 \\ + \Big[\tfrac{L}{2} - \tfrac{1}{2\eta}\Big]\|x^{t+1} - x^t\|^2 + \frac{\eta L^2}{2nb}\sum_{i=1}^{n}\|x^t - \alpha_i^t\|^2\Big].\end{aligned} \tag{5.23}$$

For further analysis, we require the following Lyapunov function:

$$R_t := \mathbb{E}\left[F(x^t) + \frac{c_t}{n}\textstyle\sum_{i=1}^{n}\|x^t - \alpha_i^t\|^2\right].$$

Moreover, for the rest of the analysis we set $\beta = b/4n$. We use p to denote the probability $1 - (1 - 1/n)^b$ of an index i being in J_t. Observe that we can bound p from below as

$$p = 1 - \left(1 - \tfrac{1}{n}\right)^b \geq 1 - \tfrac{1}{1+(b/n)} = \tfrac{b/n}{1+b/n} \geq \tfrac{b}{2n}, \tag{5.24}$$

where the first inequality follows from $(1-y)^r \leq 1/(1+ry)$ (which holds for $y \in [0,1]$ and $r \geq 1$), while the second inequality holds because $b \leq n$.

We now obtain a recursive bound on R_{t+1} as follows

$$\begin{aligned}R_{t+1} &= \mathbb{E}[F(x^{t+1}) + \frac{c_{t+1}}{n}\sum_{i=1}^{n}\|x^{t+1} - \alpha_i^{t+1}\|^2] \\ &= \mathbb{E}\big[F(x^{t+1}) + \frac{c_{t+1}p}{n}\sum_{i=1}^{n}\|x^{t+1} - x^t\|^2 + \frac{c_{t+1}(1-p)}{n}\sum_{i=1}^{n}\|x^{t+1} - \alpha_i^t\|^2\big] \\ &= \mathbb{E}\big[F(x^{t+1}) + c_{t+1}p\|x^{t+1} - x^t\|^2 \\ &\qquad + \frac{c_{t+1}(1-p)}{n}\sum_{i=1}^{n}(\|x^{t+1} - x^t\|^2 + \|x^t - \alpha_i^t\|^2 + 2\langle x^{t+1} - x^t, x^t - \alpha_i^t\rangle)\big] \\ &\leq \mathbb{E}\big[F(x^{t+1}) + c_{t+1}\big(1 + \frac{1-p}{\beta}\big)\|x^{t+1} - x^t\|^2 + \frac{c_{t+1}(1+\beta)(1-p)}{n}\sum_{i=1}^{n}\|x^t - \alpha_i^t\|^2\big] \\ &\leq \mathbb{E}\Big[F(x^t) + \Big[L - \tfrac{1}{2\eta}\Big]\|\overline{x}^{t+1} - x^t\|^2 + \Big[c_{t+1}\Big(1 + \tfrac{1-p}{\beta}\Big) + \tfrac{L}{2} - \tfrac{1}{2\eta}\Big]\|x^{t+1} - x^t\|^2 \\ &\qquad + \Big[\tfrac{c_{t+1}(1+\beta)(1-p)}{n} + \tfrac{\eta L^2}{2nb}\Big]\sum_{i=1}^{n}\|x^t - \alpha_i^t\|^2\Big] \qquad (5.25) \\ &\leq \mathbb{E}\left[F(x^t) + \Big[L - \tfrac{1}{2\eta}\Big]\|\overline{x}^{t+1} - x^t\|^2 + \Big[\tfrac{c_{t+1}(1+\beta)(1-p)}{n} + \tfrac{\eta L^2}{2nb}\Big]\sum_{i=1}^{n}\|x_t - \alpha_i^t\|^2\right] \\ &= R_t + \Big[L - \tfrac{1}{2\eta}\Big]\mathbb{E}\|\overline{x}^{t+1} - x^t\|^2. \qquad (5.26)\end{aligned}$$

The equality in the second line follows how α_i^{t+1} is chosen in Algorithm 12. In particular, from noting that each index in J_t is drawn uniformly randomly and independently from $[n]$. The first inequality follows from Cauchy-Schwarz and Young's inequality. The second inequality uses the bound (5.23). The final equality is due to the definition of the Lyapunov function R_t, wherein we also use

$$c_t = \left[c_{t+1}(1+\beta)(1-p) + \frac{\eta L^2}{2b} \right]. \tag{5.27}$$

The third inequality requires a brief explanation. It follows upon observing that

$$c_{t+1}\left(1 + \frac{1-p}{\beta}\right) + \frac{L}{2} \le \frac{1}{2\eta}. \tag{5.28}$$

To see why (5.28) holds, first observe that $c_T = 0$, and then use (5.27) to show that

$$c_t \le c_{t+1}(1-\theta) + \frac{\eta L^2}{2b},$$

where $\theta = (b/2n) - \beta = b/4n$. The above inequality is elementary, since $(1+\beta)(1-p) \le 1 - p + \beta \le (1-\theta)$ and because $p \ge (b/2n)$ as noted in (5.24). Recursing on t, we thus obtain

$$c_t \le \frac{\eta L^2}{2b}\frac{1-(1-\theta)^{T-t}}{\theta} \le \frac{2n\rho L}{b^2}, \tag{5.29}$$

for all $t \in \{0, \dots, T-1\}$, which holds due to the definition of η and θ. We now use inequality (5.29) to bound the left hand side of (5.28) as follows

$$\begin{aligned} c_{t+1}\left(1 + \frac{1-p}{\beta}\right) + \frac{L}{2} &\le \frac{2n\rho L}{b^2}\left(1 + \frac{2(2n-b)}{b}\right) + \frac{L}{2} \\ &= \frac{2n\rho L}{b^2}\left[\frac{4n}{b} - 1\right] + \frac{L}{2} \\ &\le \frac{L}{2\rho} = \frac{1}{2\eta}. \end{aligned}$$

The first inequality uses the bound (5.24), while the third inequality uses the following condition on ρ:

$$\frac{16n^2\rho^2}{b^3} + \rho \le 1.$$

Thus, inequality (5.28) holds. Adding the bound (5.26) across all the iterations and then using telescoping sums, we get

$$R_T \le R_0 + \sum_{t=0}^{T-1} \left[L - \tfrac{1}{2\eta}\right] \mathbb{E}\|\bar{x}^{t+1} - x^t\|^2. \tag{5.30}$$

Since $c_T = 0$, we observe that $R_T = \mathbb{E}[F(x^T)]$. Furthermore, since $\alpha_i^0 = x^0$ for all $i \in [n]$, we conclude that $R_0 = \mathbb{E}[F(x^0)]$. Therefore, we can rewrite (5.30) to obtain

$$\mathbb{E}[F(x^T)] \leq F(x^0) + \sum_{t=0}^{T-1} \left[L - \tfrac{1}{2\eta}\right] \mathbb{E}\|\overline{x}^{t+1} - x^t\|^2.$$

Rearranging, and using optimality of x^*, this leads to the bound

$$\sum_{t=0}^{T-1} \left[\tfrac{1}{2\eta} - L\right] \mathbb{E}\|\overline{x}^{t+1} - x^t\|^2 \leq F(x^0) - \mathbb{E}[F(x^T)] \leq F(x^0) - F(x^*).$$

Now recall the relationship

$$\mathcal{G}_\eta(x^t) = \tfrac{1}{\eta}[x^t - \text{prox}_{\eta h}(x^t - \eta \nabla f(x^t))] = \tfrac{1}{\eta}[x^t - \overline{x}^{t+1}]$$

and use the definition of x_a (from Algorithm 12) in the above bound to obtain the desired result. □

Proof of Theorem 5.3.3

Proof. The proof follows from Theorem 5.4.2 with $b = 1$ and the parameters used in the theorem statement. □

5.9.2 Proof of Theorem 5.3.4

Proof. The proof follows from Theorem 5.4.2 with $b = n^{2/3}$ and the parameters used in the theorem statement. □

5.10 Convergence Analysis of PL-variants

5.10.1 Proof of Theorem 5.5.1

Proof. The proof follows immediately from Theorem 5.10.1 and Theorem 5.10.2 with $b = n^{2/3}$ and the parameters used in theorem statement. □

5.10.2 PL-SVRG Convergence Analysis

Theorem 5.10.1. *Suppose F is a μ-PL function. Let $b \leq n$, $\eta = \rho/L$, $T = \lceil 6L/\rho\mu \rceil$, where $\rho \leq 1/5$ and satisfies the condition*

$$\frac{4\rho^2 m^2}{b} + \rho \leq 1.$$

Then for the output x^K of PL-SVRG*, the following holds:*

$$\mathbb{E}[F(x^K) - F(x^*)] \leq \frac{[F(x^0) - F(x^*)]}{2^K},$$

where x^* is an optimal solution of Problem (5.1).

Proof. We define the following :

$$\overline{x}_{t+1}^{s+1} = \text{prox}_{\eta h}(x_t^{s+1} - \eta \nabla f(x_t^{s+1})). \tag{5.31}$$

We first analyze one iteration of the PROXSVRG for PL functions. PL-SVRG essentially uses this as subroutine multiple times in order to obtain the final solution. The proof is similar to that of Theorem 5.3.4 until Equation (5.11). We have the following inequalities:

$$\mathbb{E}[F(\overline{x}_{t+1}^{s+1})] \leq \mathbb{E}\left[F(x_t^{s+1}) + \left[\frac{L}{2} - \frac{1}{\eta}\right] \|\overline{x}_{t+1}^{s+1} - x_t^{s+1}\|^2\right], \tag{5.32}$$

$$\begin{aligned}\mathbb{E}[F(x_{t+1}^{s+1})] \leq \mathbb{E}\Bigg[&F(\overline{x}_{t+1}^{s+1}) + \langle x_{t+1}^{s+1} - \overline{x}_{t+1}^{s+1}, \nabla f(x_t^{s+1}) - v_t^{s+1}\rangle \\ &+ \left[\frac{L}{2} - \frac{1}{2\eta}\right] \|x_{t+1}^{s+1} - x_t^{s+1}\|^2 + \left[\frac{L}{2} + \frac{1}{2\eta}\right] \|\overline{x}_{t+1}^{s+1} - x_t^{s+1}\|^2 - \frac{1}{2\eta}\|x_{t+1}^{s+1} - \overline{x}_{t+1}^{s+1}\|^2\Bigg].\end{aligned} \tag{5.33}$$

Furthermore, we have the following inequality:

$$\begin{aligned}\mathbb{E}[F(\overline{x}_{t+1}^{s+1})] &\leq \mathbb{E}\left[F(x_t^{s+1}) + \langle \nabla f(x_t^{s+1}, \overline{x}_{t+1}^{s+1} - x_t^{s+1}\rangle + \frac{L}{2}\|\overline{x}_{t+1}^{s+1} - x_t^{s+1}\|^2 + h(\overline{x}_{t+1}^{s+1}) - h(x_t^{s+1})\right] \\ &\leq \mathbb{E}\left[F(x_t^{s+1}) + \langle \nabla f(x_t^{s+1}, \overline{x}_{t+1}^{s+1} - x_t^{s+1}\rangle + \frac{1}{2\eta}\|\overline{x}_{t+1}^{s+1} - x_t^{s+1}\|^2 + h(\overline{x}_{t+1}^{s+1}) - h(x_t^{s+1})\right] \\ &= \mathbb{E}\left[F(x_t^{s+1}) - \frac{\eta}{2}D_h(x_t^{s+1}, \frac{1}{\eta})\right] \leq \mathbb{E}\left[F(x_t^{s+1}) - \frac{\eta}{2}D_h(x_t^{s+1}, \mu)\right] \\ &\leq \mathbb{E}\left[F(x_t^{s+1}) - \mu\eta[F(x_t^{s+1}) - F(x^*)]\right]\end{aligned} \tag{5.34}$$

The first inequality follows from Lipschitz continuity of the gradient of f. The second inequality follows from the fact that $\eta < 1/L$. The third inequality follows from the fact that $D_h(x, .)$ is a decreasing function. Here, we are implicitly using the fact that $\mu \leq L$ (which can be shown easily for μ-PL functions that are L-smooth). Adding $2/3\times$ Equation (5.32) and $1/3\times$ Equation (5.34), we have the following:

$$\mathbb{E}[F(\overline{x}_{t+1}^{s+1})] \leq \mathbb{E}\left[F(x_t^{s+1}) + \left[\frac{L}{3} - \frac{2}{3\eta}\right] \|\overline{x}_{t+1}^{s+1} - x_t^{s+1}\|^2 - \frac{\mu\eta}{3}[F(x_t^{s+1}) - F(x^*)]\right]. \tag{5.35}$$

Adding the above equation with Equation (5.33), we have the following:

$$\begin{aligned}\mathbb{E}[F(x_{t+1}^{s+1})] \leq \mathbb{E}\Big[&F(x_t^{s+1}) + \left[\frac{5L}{6} - \frac{1}{6\eta}\right] \|\overline{x}_{t+1}^{s+1} - x_t^{s+1}\|^2 + \left[\frac{L}{2} - \frac{1}{2\eta}\right] \|x_{t+1}^{s+1} - x_t^{s+1}\|^2 \\ &- \frac{\mu\eta}{3}[F(x_t^{s+1}) - F(x^*)] - \frac{1}{2\eta}\|x_{t+1}^{s+1} - \overline{x}_{t+1}^{s+1}\|^2 + \langle x_{t+1}^{s+1} - \overline{x}_{t+1}^{s+1}, \nabla f(x_t^{s+1}) - v_t^{s+1}\rangle\Big].\end{aligned} \tag{5.36}$$

Using Cauchy-Schwarz and Young's inequality and the fact that $\eta \leq 1/5L$, we have the following:

$$\begin{aligned}
&\mathbb{E}[F(x_{t+1}^{s+1})] && (5.37)\\
&\leq \mathbb{E}\left[F(x_t^{s+1}) + \left[\tfrac{L}{2} - \tfrac{1}{2\eta}\right]\|x_{t+1}^{s+1} - x_t^{s+1}\|^2 - \tfrac{\mu\eta}{3}[F(x_t^{s+1}) - F(x^*)] + \tfrac{\eta}{2}\|\nabla f(x_t^{s+1}) - v_t^{s+1}\|^2\right]\\
&\leq \mathbb{E}\left[F(x_t^{s+1}) + \left[\tfrac{L}{2} - \tfrac{1}{2\eta}\right]\|x_{t+1}^{s+1} - x_t^{s+1}\|^2 - \tfrac{\mu\eta}{3}[F(x_t^{s+1}) - F(x^*)] + \tfrac{\eta L^2}{2b}\|x_t^{s+1} - \tilde{x}^s\|^2\right]. && (5.38)
\end{aligned}$$

The second inequality follows from Lemma 5.12.3. We use the similar proof technique as in Theorem 5.4.1 and 5.4.2 and define the following lyapunov function: $R_{t+1}^{s+1} = \mathbb{E}[F(x_{t+1}^{s+1}) + c_{t+1}\|x_{t+1}^{s+1} - \tilde{x}^s\|^2]$. Let $\beta = b/n$. Using the bound on the lyapunov function in Equation (5.15), we have the following:

$$\begin{aligned}
R_{t+1}^{s+1} &\leq \mathbb{E}\Big[F(x_t^{s+1}) - \tfrac{\mu\eta}{3}[F(x_t^{s+1}) - F(x^*)] + \left[c_{t+1}\left(1 + \tfrac{1}{\beta}\right) + \tfrac{L}{2} - \tfrac{1}{2\eta}\right]\|x_{t+1}^{s+1} - x_t^{s+1}\|^2\\
&\qquad + \left[c_{t+1}(1+\beta) + \tfrac{\eta L^2}{2b}\right]\|x_t^{s+1} - \tilde{x}^s\|^2\Big]\\
&\leq \mathbb{E}\left[F(x_t^{s+1}) - \tfrac{\mu\eta}{3}[F(x_t^{s+1}) - F(x^*)] + \left[c_{t+1}(1+\beta) + \tfrac{\eta L^2}{2b}\right]\|x_t^{s+1} - \tilde{x}^s\|^2\right]\\
&= R_t^{s+1} - \tfrac{\mu\eta}{3}\mathbb{E}[F(x_t^{s+1}) - F(x^*)]. && (5.39)
\end{aligned}$$

The second inequality follows from the fact that:

$$c_{t+1}\left(1 + \frac{1}{\beta}\right) + \frac{L}{2} \leq \frac{1}{2\eta}.$$

This, again, follows from argument stated in Theorem 5.4.1, the fact that $\eta = \rho/L$ and

$$\frac{4\rho^2 m^2}{b} + \rho \leq 1.$$

Adding Equation (5.39) across all the iterations epoch $s+1$ and then using telescopy sum, we get

$$R_m^{s+1} \leq R_0^{s+1} - \sum_{t=0}^{m-1} \frac{\mu\eta}{3}\mathbb{E}[F(x_t^{s+1}) - F(x^*)]. \tag{5.40}$$

We observe that $R_m^{s+1} = \mathbb{E}[F(x_m^{s+1})] = \mathbb{E}[F(\tilde{x}^{s+1})]$. This is due the fact that $c_m = 0$ and the definition of $\tilde{x}^{s+1}$. Furthermore, $R_0^{s+1} = \mathbb{E}[F(x_0^{s+1})] = \mathbb{E}[F(\tilde{x}^s)]$. This is due to the fact that $x_0^{s+1} = \tilde{x}^s$. Therefore, using the above reasoning in Equation (5.40), we have

$$\mathbb{E}[F(\tilde{x}^{s+1})] \leq \mathbb{E}[F(\tilde{x}^s)] - \sum_{t=0}^{m-1} \frac{\mu\eta}{3}\mathbb{E}[F(x_t^{s+1}) - F(x^*)].$$

Adding the inequality stated above across all the epochs and using telescopy sum, we have:

$$\sum_{s=0}^{S}\sum_{t=0}^{m-1}\frac{\mu\eta}{3}\mathbb{E}[F(x_t^{s+1})-F(x^*)] \le \mathbb{E}[F(x^0)] - \mathbb{E}[F(\tilde{x}^S)] \le \mathbb{E}[F(x^0)] - F(x^*).$$

The second inequality follows from the optimality of x^*. Using the definition of x^k in PL-SVRG, we have the following:

$$\begin{aligned}\mathbb{E}[F(x^1)-F(x^*)] &\le \frac{3\mathbb{E}[F(x^0)-F(x^*)]}{\mu\eta T}\\ &\le \frac{\mathbb{E}[F(x^0)-F(x^*)]}{2}.\end{aligned}$$

The second inequality follows from the fact that $T = \lceil 6L/\rho\mu \rceil$. Using this recursion, we have the desired result. □

5.10.3 PL-SAGA Convergence Analysis

Theorem 5.10.2. *Suppose F is a μ-PL function. Let $b \le n$, $\eta = \rho/L$, $T = \lceil 6L/\mu\rho \rceil$ where $\rho \le 1/5$ and satisfies the following condition:*

$$\frac{16n^2\rho^2}{b^3} + \rho \le 1.$$

Then for the output x^K of PL-SAGA, the following holds:

$$\mathbb{E}[F(x^K)-F(x^*)] \le \frac{[F(x^0)-F(x^*)]}{2^K},$$

where x^* is an optimal solution of Problem (5.1).

Proof. We define the following :

$$\overline{x}^{t+1} = \text{prox}_{\eta h}(x^t - \eta\nabla f(x^t)). \tag{5.41}$$

Similar to Theorem 5.10.1, we start with one iteration of PL-SAGA algorithm. In particular, we first analyze the case of T iterations of SAGA. Further recursing on the the result obtain will give us the desired result. The first part of the theorem is similar to the proof in Theorem 5.10.1. Using essentially a similar argument as the one in Theorem 5.10.1 until Equation (5.32), we have the following:

$$\mathbb{E}[F(x^{t+1})] \le \mathbb{E}\left[F(x^t) + \left[\frac{L}{2}-\frac{1}{2\eta}\right]\|x^{t+1}-x^t\|^2 - \frac{\mu\eta}{3}[F(x^t)-F(x^*)] + \frac{\eta L^2}{2nb}\sum_{i=1}^{n}\|x^t-\alpha_i^t\|^2\right]. \tag{5.42}$$

We the following Lyapunov function:

$$R_t = \mathbb{E}[F(x^t) + \frac{c_t}{n}\sum_{i=1}^{n}\|x^t - \alpha_i^t\|^2],$$

as defined in Theorem 5.4.2. Using the same argument in Theorem 5.4.2 to bound it, we have the following:

$$\begin{aligned}
R_{t+1} &\le \mathbb{E}\Big[F(x^t) - \tfrac{\mu\eta}{3}[F(x^t) - F(x^*)] + \Big[c_{t+1}\Big(1 + \tfrac{1-p}{\beta}\Big) + \tfrac{L}{2} - \tfrac{1}{2\eta}\Big]\|x^{t+1} - x^t\|^2 \\
&\qquad + \Big[\tfrac{c_{t+1}(1+\beta)(1-p)}{n} + \tfrac{\eta L^2}{2nb}\Big]\textstyle\sum_{i=1}^{n}\|x^t - \alpha_i^t\|^2\Big] \\
&\le R_t - \tfrac{\mu\eta}{3}\mathbb{E}[F(x^t) - F(x^*)]. \qquad (5.43)
\end{aligned}$$

Recall that $p = 1 - (1 - 1/n)^b$. The second inequality is due to the following inequality:

$$c_{t+1}\left(1 + \frac{1-p}{\beta}\right) + \frac{L}{2} \le \frac{1}{2\eta}.$$

This is obtained by the same argument in Theorem 5.4.2. Adding Equation (5.43) over all the iterations and using telescopy sum, we have the following:

$$\mathbb{E}[F(x^T)] \le \mathbb{E}[F(x^0)] - \sum_{t=0}^{T-1}\frac{\mu\eta}{3}\mathbb{E}[F(x^t) - F(x^*)].$$

The above inequality is obtained from the fact that $R_T = \mathbb{E}[F(x^T)]$. This is due the fact that $c_T = 0$. Furthermore, $R_0 = \mathbb{E}[F(x^0)]$. This is due to the fact that $\alpha_i^0 = x^0$ for all $i \in [n]$. Therefore, we have:

$$\sum_{t=0}^{T-1}\frac{\mu\eta}{3}\mathbb{E}[F(x^t) - F(x^*)] \le \mathbb{E}[F(x^0)] - \mathbb{E}[F(x^T)] \le \mathbb{E}[F(x^0) - F(x^*)].$$

Using the definition of x^k in PL-SAGA, we have the following:

$$\begin{aligned}
\mathbb{E}[F(x^1) - F(x^*)] &\le \frac{3\mathbb{E}[F(x^0) - F(x^*)]}{\mu\eta T} \\
&\le \frac{\mathbb{E}[F(x^0) - F(x^*)]}{2}.
\end{aligned}$$

The second inequality follows from the fact that $T = \lceil 6L/\mu\rho \rceil$. Using the above recursion repeatedly, we obtain the desired result. □

5.11 Additional Experiments

We present the additional experiments for non-negative principal component analysis problems in this section. Figure 5.3 shows the additional results. Similar to Figure 5.2, we see that PROXSVRG and PROXSAGA outperform PROXSGD. We did not find any significant difference in the performance of PROXSVRG and PROXSAGA.

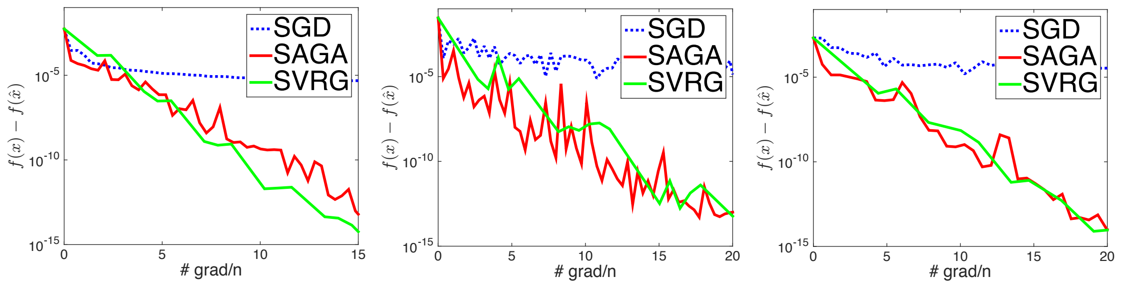


Figure 5.3: Non-negative principal component analysis. Performance of SGD, PROXSVRG and PROXSAGA on 'real-sim' (left), 'covtype'(center) and 'ijcnn1' (right) datasets. Recall that the y-axis is the function suboptimality i.e., $f(x) - f(\hat{x})$ where $\hat{x}$ represents the best solution obtained by running gradient descent for long time and with multiple restarts.

5.12 Lemmatta

We first few intermediate results that are useful for our analysis. These results are key to the mirror descent analysis [108]. We prove them here for completeness.

Lemma 5.12.1. *Suppose we define the following:*

$$y = \text{prox}_{\eta h}(x - \eta d'). \tag{5.44}$$

for some $d' \in \mathbb{R}^d$. Then for y, the following inequality holds:

$$h(y) + \langle y - z, d' \rangle \leq h(z) + \frac{1}{2\eta}\left[\|z - x\|^2 - \|y - x\|^2 - \|y - z\|^2\right]. \tag{5.45}$$

for all $z \in \mathbb{R}^d$.

Proof. From Lemma 5.12.5 applied on Equation (5.44), we get the following:

$$\begin{aligned}
&h(y) + \langle y - x, d' \rangle + \frac{1}{2\eta}\|y - x\|^2 + \frac{\eta}{2}\|d'\|^2 \\
&= h(y) + \frac{1}{2\eta}\|y - (x - \eta d')\|^2 \\
&\leq h(z) + \frac{1}{2\eta}\|z - (x - \eta d')\|^2 - \frac{1}{2\eta}\|y - z\|^2 \\
&= h(z) + \langle z - x, d' \rangle + \frac{1}{2\eta}\|z - x\|^2 + \frac{\eta}{2}\|d'\|^2 - \frac{1}{2\eta}\|y - z\|^2.
\end{aligned} \tag{5.46}$$

By rearranging Equation (5.46), we obtain the following inequality that concludes the proof.

$$h(y) + \langle y - z, d' \rangle \leq h(z) + \frac{1}{2\eta}\left[\|z - x\|^2 - \|y - x\|^2 - \|y - z\|^2\right].$$

□

The following key lemma involving function F is useful for proving the convergence of ProxSvrg and ProxSaga.

Lemma 5.12.2. *Suppose we define the following:*

$$y = \mathrm{prox}_{\eta h}(x - \eta d').$$

for some $d' \in \mathbb{R}^d$. Then for y, the following inequality holds:

$$\begin{aligned} F(y) \leq F(z) + & \langle y - z, \nabla f(x) - d' \rangle \\ & + \left[\tfrac{L}{2} - \tfrac{1}{2\eta}\right] \|y - x\|^2 + \left[\tfrac{L}{2} + \tfrac{1}{2\eta}\right] \|z - x\|^2 - \tfrac{1}{2\eta}\|y - z\|^2. \end{aligned} \tag{5.47}$$

for all $z \in \mathbb{R}^d$.

Proof. We have the following inequalities for function f:

$$f(y) \leq f(x) + \langle \nabla f(x), y - x \rangle + \frac{L}{2}\|y - x\|^2,$$
$$f(x) \leq f(z) + \langle \nabla f(x), x - z \rangle + \frac{L}{2}\|x - z\|^2.$$

The above inequalities can be obtained by application of Lemma 5.12.6. Adding both the inequalities above, we obtain the following inequality:

$$f(y) \leq f(z) + \langle \nabla f(x), y - z \rangle + \tfrac{L}{2}\left[\|y - x\|^2 + \|z - x\|^2\right]. \tag{5.48}$$

Adding Equations (5.45) (which follows from Lemma 5.12.1) and (5.48), we obtain the inequality:

$$\begin{aligned} F(y) \leq F(z) + & \langle y - z, \nabla f(x) - d' \rangle \\ & + \left[\tfrac{L}{2} - \tfrac{1}{2\eta}\right] \|y - x\|^2 + \left[\tfrac{L}{2} + \tfrac{1}{2\eta}\right] \|z - x\|^2 - \tfrac{1}{2\eta}\|y - z\|^2. \end{aligned} \tag{5.49}$$

Here we used that definition $F(x) = f(x) + h(x)$. This concludes our proof. □

The following result is useful for bounding the variance of the updates of ProxSvrg and follows from slight modification of result in [139]. We give the proof here for completeness.

Lemma 5.12.3 ([139])**.** *For the iterates x_t^{s+1}, v_t^{s+1} and $\tilde{x}^s$ where $t \in \{0, \dots, m-1\}$ and $s \in \{0, \dots, S-1\}$ in Algorithm 11, the following inequality holds:*

$$\mathbb{E}[\|\nabla f(x_t^{s+1}) - v_t^{s+1}\|^2] \leq \frac{L^2}{b}\|x_t^{s+1} - \tilde{x}^s\|^2.$$

Proof. Let us define the following notation for the ease of exposition:

$$\zeta_t^{s+1} = \frac{1}{|I_t|}\sum_{i\in I_t}\left(\nabla f_i(x_t^{s+1}) - \nabla f_i(\tilde{x}^s)\right).$$

Using this notation, we obtain the following bound:

$$\begin{aligned}
&\mathbb{E}[\|\nabla f(x_t^{s+1}) - v_t^{s+1}\|^2] = \mathbb{E}[\|\zeta_t^{s+1} + \nabla f(\tilde{x}^s) - \nabla f(x_t^{s+1})\|^2] \\
&= \mathbb{E}[\|\zeta_t^{s+1} - \mathbb{E}[\zeta_t^{s+1}]\|^2] = \frac{1}{b^2}\mathbb{E}\left[\left\|\sum_{i\in I_t}\left(\nabla f_i(x_t^{s+1}) - \nabla f_i(\tilde{x}^s) - \mathbb{E}[\zeta_t^{s+1}]\right)\right\|^2\right]
\end{aligned}$$

The second equality is due to the fact that $\mathbb{E}[\zeta_t^{s+1}] = \nabla f(x_t^{s+1}) - \nabla f(\tilde{x}^s)$. From the above relationship, we get

$$\begin{aligned}
\mathbb{E}[\|\nabla f(x_t^{s+1}) - v_t^{s+1}\|^2] &\leq \frac{1}{b}\mathbb{E}\left[\sum_{i\in I_t}\|\nabla f_i(x_t^{s+1}) - \nabla f_i(\tilde{x}^s) - \mathbb{E}[\zeta_t^{s+1}]\|^2\right] \\
&\leq \frac{1}{b}\mathbb{E}\left[\sum_{i\in I_t}\|\nabla f_i(x_t^{s+1}) - \nabla f_i(\tilde{x}^s)\|^2\right] \leq \frac{L^2}{b}\mathbb{E}[\|x_t^{s+1} - \tilde{x}^s\|^2]
\end{aligned}$$

The first inequality follows from Lemma 5.12.7. The second inequality is due to the fact that for a random variable ζ, $\mathbb{E}[\|\zeta - \mathbb{E}[\zeta]\|^2] \leq \mathbb{E}[\|\zeta\|^2]$. The last inequality follows from L-smoothness of f_i. □

A similar result can be obtained for ProxSaga. The key difference with that of Lemma 5.12.3 is that the variance term in ProxSaga involves α_i^t. Again, we provide the proof for completeness.

Lemma 5.12.4. *For the iterates x^t, v^t and $\{\alpha_i^t\}_{i=1}^n$ where $t \in \{0,\dots,T-1\}$ in Algorithm 12, the following inequality holds:*

$$\mathbb{E}[\|\nabla f(x^t) - v^t\|^2] \leq \frac{L^2}{nb}\sum_{i=1}^n \mathbb{E}\|x^t - \alpha_i^t\|^2.$$

Proof. Let us define the following notation for the ease of exposition:

$$\zeta_t = \frac{1}{|I_t|}\sum_{i\in I_t}\left(\nabla f_i(x^t) - \nabla f_i(\alpha_i^t)\right).$$

In this notation, we have the following:

$$\begin{aligned}
&\mathbb{E}[\|\nabla f(x^t) - v^t\|^2] = \mathbb{E}\left[\left\|\zeta_t + \frac{1}{n}\sum_{i=1}^n \nabla f(\alpha_i^t) - \nabla f(x^t)\right\|^2\right] \\
&= \mathbb{E}[\|\zeta_t - \mathbb{E}[\zeta_t]\|^2] = \frac{1}{b^2}\mathbb{E}\left[\left\|\sum_{i\in I_t}\left(\nabla f_i(x^t) - \nabla f_i(\alpha_i^t) - \mathbb{E}[\zeta_t]\right)\right\|^2\right]
\end{aligned}$$

The second equality follows from the fact that $\mathbb{E}[\zeta_t] = \nabla f(x^t) - \frac{1}{n}\sum_{i=1}^{n} \nabla f(\alpha_i^t)$. From the above inequality, we get

$$\begin{aligned}\mathbb{E}[\|\nabla f(x^t) - v^t\|^2] &\leq \frac{1}{b}\mathbb{E}\left[\sum_{i\in I_t} \|\nabla f_i(x^t) - \nabla f_i(\alpha_i^t) - \mathbb{E}[\zeta_t]\|^2\right] \\ &\leq \frac{1}{b}\mathbb{E}\left[\sum_{i\in I_t} \|\nabla f_i(x^t) - \nabla f_i(\alpha_i^t)\|^2\right] \leq \frac{L^2}{nb}\sum_{i=1}^{n} \mathbb{E}[\|x^t - \alpha_i^t\|^2]\end{aligned}$$

The first inequality is due to Lemma 5.12.7. The second inequality follows from the fact that for a random variable ζ, $\mathbb{E}[\|\zeta - \mathbb{E}[\zeta]\|^2] \leq \mathbb{E}[\|\zeta\|^2]$. The last inequality is from L-smoothness of f_i for all $i \in [n]$ and uniform randomness of the set I_t. □

The following lemma is a classical result in mirror descent analysis.

Lemma 5.12.5. *Suppose function $h : \mathbb{R}^d \to \mathbb{R}$ is l.s.c and $y = \operatorname{prox}_{\eta h}(x)$. Then we have the following:*

$$h(y) + \tfrac{1}{2\eta}\|y - x\|^2 \leq h(z) + \tfrac{1}{2\eta}\|z - x\|^2 - \tfrac{1}{2\eta}\|y - z\|^2,$$

for all $z \in \mathbb{R}^d$.

Lemma 5.12.6. *Suppose function $f : \mathbb{R}^d \to \mathbb{R}$ is L-smooth, then we have the following:*

$$f(y) + \langle \nabla f(y), x - y\rangle - \frac{L}{2}\|x - y\|^2 \leq f(x) \leq f(y) + \langle \nabla f(y), x - y\rangle + \frac{L}{2}\|x - y\|^2,$$

for all $x, y \in \mathbb{R}^d$.

Lemma 5.12.7. *For random variables $z_1, \dots, z_r$ are independent and mean 0, we have*

$$\mathbb{E}\left[\|z_1 + ... + z_r\|^2\right] = \mathbb{E}\left[\|z_1\|^2 + ... + \|z_r\|^2\right].$$

Proof. We have the following:

$$\mathbb{E}\left[\|z_1 + ... + z_r\|^2\right] = \sum_{i,i=1}^{r} \mathbb{E}\left[z_i z_j\right] = \mathbb{E}\left[\|z_1\|^2 + ... + \|z_r\|^2\right].$$

The second equality follows from the fact that z_i's are independent and mean 0. □

Lemma 5.12.8. *For random variables $z_1, \dots, z_r$, we have*

$$\mathbb{E}\left[\|z_1 + ... + z_r\|^2\right] \leq r\mathbb{E}\left[\|z_1\|^2 + ... + \|z_r\|^2\right].$$

Chapter 6

Projection-Free Stochastic Nonconvex Optimization

6.1 Introduction

This chapter focuses on designing projection-free methods for constrained optimization problems of the form:

$$\min_{x \in \Omega} F(x) := \begin{cases} \mathbb{E}_z[f(x,z)], & \text{(stochastic)} \\ \frac{1}{n}\sum_{i=}^{n} f_i(x), & \text{(finite-sum).} \end{cases} \tag{6.1}$$

We assume that F, f, and f_i ($i \in \{1, \ldots, n\} \triangleq [n]$) are all differentiable, but possibly *nonconvex*; the domain Ω is *convex* and *compact*. Problems of this form are at the heart of machine learning and statistics. Examples of such problems include multiclass classification, matrix learning, recommendation systems [54, 55, 56, 66].

Within convex optimization, problem (6.1) is relatively well-studied. Two particularly popular approaches for solving it are: (i) Projected stochastic gradient descent (SGD); and (b) the Frank-Wolfe (FW) method. At each iteration, SGD takes a step in a direction opposite to a stochastic approximation of the gradient ∇F and uses projection onto Ω to ensure feasibility. While computing a stochastic approximation to ∇F is usually inexpensive, in many real settings, the cost projecting onto Ω can be very high (e.g., projecting onto the trace-norm ball, onto base polytopes in submodular minimization [42]); and in extreme cases projection can even be computationally intractable [28].

In such cases, projection based methods like SGD become impractical. This difficulty underlies the recent surge of interest in Frank-Wolfe methods [40, 66] (also known as conditional gradient), due to their projection-free property. In particular, FW methods avoid the expensive projection operation and require just a *linear oracle* that solves problems of the form $\min_{x \in \Omega} \langle x, g \rangle$ at each iteration.

Despite the remarkable success of FW approaches in the convex setting, including stochastic problems [56], their applicability and *non-asymptotic convergence* for *nonconvex* optimization is largely unstudied. Even for SGD, it is only recently that non-asymptotic convergence analysis for nonconvex optimization was obtained [47, 48]. More recently, Reddi et al. [139, 141] obtained variance reduced stochastic methods that converge faster than SGD in the *nonconvex* finite-sum setting.

Similar fast variants of FW for nonconvex problems are not known. Given the vast importance of nonconvex models in machine learning (e.g., in deep learning) and the need to incorporate non-trivial constraints in such models, it is imperative to develop scalable, projection-free methods. This chapter presents new FW methods towards this goal. Our main contributions are summarized below, while the key complexity results are listed in Figure 6.1.

Main Contributions. For the nonconvex stochastic setting in (6.1), we propose a stochastic Frank-Wolfe algorithm (SFW), and provide its convergence analysis. For the nonconvex finite-sum setting, we propose two variance reduced (VR) algorithms: SVFW and SAGAFW, based on the popular VR algorithms SVRG and SAGA, respectively. We show that by carefully selecting the parameters of these algorithms, we can attain faster convergence rates than the deterministic FW. In particular, we prove that SVFW and SAGAFW are faster than deterministic FW by a factor of $n^{1/3}$ and $n^{2/3}$ respectively, where n is the number of component functions in the finite-sum (see (6.1)). Furthermore, leveraging these variance reduced methods, we propose two algorithms, SVFW-S and SAGAFW-S, for the nonconvex stochastic setting, with faster convergence rates than SFW. To our knowledge, our work presents the first theoretical improvement for stochastic variants of Frank-Wolfe in the context of nonconvex optimization.

6.1.1 Related Work

The classical Frank-Wolfe method [40] using line-search was analyzed for smooth convex functions F and polyhedral domains Ω. Here, a convergence rate of $O(1/\epsilon)$ to ensure $F(x) - F^* \leq \epsilon$ was proved without additional conditions [40, 66]. There have been several recent works on improving the convergence rates under additional assumptions [45, 79]. More recently, Hazan and Luo [56] proposed stochastic variants of FW for convex problems of form (6.1), and showed theoretical improvements over the classical Frank-Wolfe method.

The literature on nonconvex Frank-Wolfe is relatively small. The work [16] proves asymptotic convergence of FW to a stationary point; though, no convergence rates are provided. To the best of our knowledge, Yu et al. [178] is the first to provide convergence rates for FW-type algorithm in the nonconvex setting. Very recently, Lacoste-Julien [78] provided a (non-asymptotic) convergence rate of $O(1/\epsilon^2)$ for nonconvex FW with adaptive step sizes. However, as we shall see later, implementation of classical FW for (6.1) is expensive (or impossible in the pure stochastic case) since it requires calculation of the gradient ∇F at each iteration. We show that our stochastic variants are provably faster than the existing FW methods.

In the nonconvex setting, most of the work on stochastic methods focuses on SGD [47, 48] and analyzes convergence to stationary points. For the finite-sum setting, we build on recent variance reduction techniques [33, 71, 153], which were first proposed for solving unconstrained convex problems of form (6.1). Projected variants to handle constraints were studied in [33, 173]. More recently, Reddi et al. [139, 141, 142] provided nonconvex variants of these methods that converge provably faster than both SGD and its deterministic counterpart.

6.2 Preliminaries

As stated above, we study two different problem settings: (1) *stochastic*, where $F(x) = \mathbb{E}_z[f(x,z)]$ and z is random variable whose distribution $\mathcal{P}$ is supported on $\Xi \subset \mathbb{R}^p$; and (2) *finite-sums*, where $F(x) = \frac{1}{n}\sum_{i=1}^n f_i(x)$.

For the stochastic setting, we assume that F is *L-smooth*, i.e., its gradient is Lipschitz continuous with constant L, so

$$\|\nabla F(x) - \nabla F(y)\| \leq L\|x-y\|, \quad \forall\, x, y \in \Omega.$$

Here $\|.\|$ denotes the ℓ^2-norm. Furthermore, for the stochastic setting, we also assume the function f is G-Lipschitz i.e., $\|\nabla f(x,z)\| \leq G$ for all $x \in \Omega$ and $z \in \Xi$. Such an assumption is common in the stochastic setting [47, 56].

For the finite-sum setting, we assume that the individual functions f_i ($i \in [n]$) are L-smooth i.e.,

$$\|\nabla f_i(x) - \nabla f_i(y)\| \leq L\|x-y\|, \quad \forall\, x, y \in \Omega.$$

Note that this implies that the function F is also L-smooth. The domain $\Omega \in \mathbb{R}^d$ is assumed to be convex and compact with diameter D; i.e., $\|x-y\| \leq D$ for all $x, y \in \Omega$. Such an assumption is common to all Frank-Wolfe methods.

Convergence criteria. The criterion used for the convergence analysis is important in nonconvex optimization. For unconstrained problems, the gradient norm $\|\nabla F\|$ is typically used to measure convergence, because $\|\nabla F\| \to 0$ translates into convergence to a stationary point. However, this criterion cannot be used for constrained problems of the form (6.1). Instead, we use the following quantity, typically referred to as *Frank-Wolfe gap*:

$$\mathcal{G}(x) = \max_{v \in \Omega} \langle v - x, -\nabla F(x) \rangle. \tag{6.2}$$

For convex functions, the FW gap provides an upper bound on the suboptimality. For nonconvex functions, the gap $\mathcal{G}(x) = 0$ if and only if x is a stationary point. To state our convergence results we will also need the following bound:

$$\beta \geq \frac{2(F(x_0) - F(x^*))}{LD^2},$$

given some (unspecified) initial point $x_0 \in \Omega$.

Oracle model. To compare convergence speed of different algorithms, we use the following black-box oracles:

- *Stochastic First-Order Oracle* (SFO): For a function $F(\cdot) = \mathbb{E}_z[f(.,z)]$ where $z \sim \mathcal{P}$, an SFO takes a point x and returns the pair $(f(x,z'), \nabla f(x,z'))$ where z' is a sample drawn i.i.d. from $\mathcal{P}$ [108].

Algorithm	SFO/IFO Complexity	LO Complexity
Frank-Wolfe	$O(n/\epsilon^2)$	$O(1/\epsilon^2)$
SFW	$O(1/\epsilon^4)$	$O(1/\epsilon^2)$
SVFW	$O(n + n^{2/3}/\epsilon^2)$	$O(1/\epsilon^2)$
SAGAFW	$O(n + n^{1/3}/\epsilon^2)$	$O(1/\epsilon^2)$
SVFW-S	$O(1/\epsilon^{10/3})$	$O(1/\epsilon^2)$
SAGAFW-S	$O(1/\epsilon^{8/3})$	$O(1/\epsilon^2)$

Figure 6.1: Table comparing the *best* SFO/IFO and LO complexity of algorithms discussed in the chapter (for the nonconvex setting). Here, SFW, SVFW-S and SAGAFW-S are algorithms for the stochastic setting, while FW, SVFW and SAGAFW are algorithms for the finite-sum setting. The complexity is measured by the number of oracle calls required to achieve an ϵ-accurate solution (see Section 6.2 for definitions of SFO/IFO and LO complexity). The complexity of FW is from [78]. The results marked in red highlight our contributions in this chapter. For clarity, we hide the dependence of SFO/IFO and LO complexity on the initial point and few parameters related to the function F and domain Ω.

- *Incremental First-Order Oracle* (IFO): For a function $F(\cdot) = \frac{1}{n}\sum_i f_i(.)$, an IFO takes an index $i \in [n]$ and a point $x \in \mathbb{R}^d$, and returns the pair $(f_i(x), \nabla f_i(x))$ [2].
- *Linear Optimization Oracle* (LO): For a set Ω, an LO takes a direction d and returns $\arg\max_{v\in\Omega}\langle v, d\rangle$.

Throughout the chapter, by SFO, IFO and LO complexity of an algorithm, we mean the total number of SFO, IFO and LO calls made by the algorithm to obtain an *ϵ-accurate solution*, i.e., a solution for which $\mathbb{E}[\mathcal{G}(x)] \leq \epsilon$; the expectation is over any randomization as part of the algorithm. For clarity of presentation, we hide the dependence of these complexities on the initial point $F(x_0) - F(x^*)$, Lipschitz constant G, and the smoothness constant L; we report the dependence on n to highlight its importance.

Classical FW. To place our results in perspective, we begin by recalling the classical Frank-Wolfe (FW) algorithm [40]. Pseudocode for this is presented in Algorithm 13.

Algorithm 13: FW $\left(x_0, T, \{\gamma_i\}_{i=0}^{T-1}\right)$

```
1: Input: $x_0 \in \Omega$, number of iterations $T$, $\{\gamma_i\}_{i=0}^{T-1}$ where $\gamma_i \in [0,1]$ for all $i \in \{0,\dots,T-1\}$
2: for $t = 0$ to $T-1$ do
3:     Compute $v_t = \arg\max_{v\in\Omega}\langle v, -\nabla F(x_t)\rangle$
4:     Compute update direction $d_t = v_t - x_t$
5:     $x_{t+1} = x_t + \gamma_t d_t$
6: end for
```

Each iteration of FW entails calculation of the gradient ∇F and moving towards a minimizer of a linearized objective. Notice that calculation of ∇F may not be possible in the stochastic setting of (6.1). Furthermore, even in the finite-sum setting, computing ∇F requires n IFO calls, rendering the approach useless in large-scale problems, where n is large. For the nonconvex finite-sum setting, the following key result was proved recently [78].

Theorem 6.2.1 ([78]). *Under appropriate selection of step sizes γ_t, the IFO and LO complexity of Algorithm 13 to achieve an ϵ-accurate solution in the finite-sum setting are $O(n/\epsilon^2)$ and $O(1/\epsilon^2)$ respectively.*

The key aspect of Theorem 6.2.1 is the dependence of IFO complexity on n. In particular, when n is large, the IFO complexity $O(n/\epsilon^2)$ shown by the theorem becomes prohibitively expensive; thus, undermining the benefits of FW over competitors like projected SGD. In the next section, we tackle this drawback and develop faster nonconvex stochastic and variance reduced FW methods.

6.3 Algorithms

In this section, we describe FW algorithms for solving (6.1). In particular, we explore stochastic and variance reduced versions of the classical FW method, for the stochastic and finite-sum settings, respectively. We defer the discussion on comparison of the convergence rates to Section 6.5.

6.3.1 Stochastic Setting

We first investigate the convergence of FW in the stochastic setting. As mentioned earlier, the classical FW method (Algorithm 13) requires calculation of the full gradient $\nabla F(x)$, which is typically impossible to compute in the stochastic setting. For convex problems, Hazan and Luo [56] tackle this issue by using the popular Robbins-Monro approximation [149] to the gradient. We use a variant of the algorithm for our nonconvex stochastic setting, which we call SFW.

The pseudocode of SFW is listed in Algorithm 14. Note that the samples $\{z_i\}$ are chosen independently according to the distribution $\mathcal{P}$. Thus, $\mathbb{E}_{z_i}[\nabla f(x, z_i)] = \nabla F(x)$, i.e., we obtain an unbiased estimate of the gradient. Also, note that the output in Algorithm 14 is randomly selected from all the iterates of the algorithm. The key parameters of SFW are the step sizes $\{\gamma_i\}_{i=0}^{T-1}$ and the minibatch sizes $\{b_t\}$. These parameters must be chosen appropriately in order to ensure convergence of the algorithm (see Theorem 6.3.1). For our analysis, we assume that the function f is G-Lipschitz i.e., we have $\max_{x\in\Omega, z\in\Xi} \|\nabla f(x,z)\| \leq G$. This bound on the gradient is crucial for our convergence analysis.

We prove the following key result for nonconvex SFW.

Algorithm 14: Nonconvex SFW $\left(x_0, T, \{\gamma_i\}_{i=0}^{T-1}, \{b_i\}_{i=0}^{T-1}\right)$

```
1: Input: $x_0 \in \Omega$, number of iterations $T$, $\{\gamma_i\}_{i=0}^{T-1}$ where $\gamma_i \in [0,1]$ for all
   $i \in \{0, \dots, T-1\}$, minibatch size $\{b_i\}_{i=0}^{T-1}$
2: for $t = 0$ to $T-1$ do
3:     Uniformly randomly pick i.i.d samples $\{z_1^t, \dots, z_{b_t}^t\}$ according to the distribution
       $\mathcal{P}$.
4:     Compute $v_t = \arg\max_{v \in \Omega} \langle v, -\frac{1}{b_t} \sum_{i=1}^{b_t} \nabla f(x_t, z_i) \rangle$
5:     Compute update direction $d_t = v_t - x_t$
6:     $x_{t+1} = x_t + \gamma_t d_t$
7: end for
8: Output: Iterate $x_a$ chosen uniformly random from $\{x_t\}_{t=0}^{T-1}$.
```

Theorem 6.3.1. *Consider the stochastic setting of* (6.1) *where f is G-Lipschitz and F is L-smooth. Then, the output x_a of Algorithm 14 with parameters $\gamma_t = \gamma = \sqrt{\frac{2(F(x_0)-F(x^*))}{TLD^2\beta}}$, $b_t = b = T$ for all $t \in \{0, \dots, T-1\}$, satisfies the following bound:*

$$\mathbb{E}[\mathcal{G}(x_a)] \leq \frac{D}{\sqrt{T}} \left(G + \sqrt{\frac{2L(F(x_0)-F(x^*))}{\beta}} (1+\beta) \right),$$

where x^* is an optimal solution to (stochastic) problem (6.1).

Proof. First observe the following upper bound:

$$\begin{aligned} F(x_{t+1}) &\leq F(x_t) + \langle \nabla F(x_t), x_{t+1} - x_t \rangle + \frac{L}{2} \|x_{t+1} - x_t\|^2 \\ &= F(x_t) + \langle \nabla F(x_t), \gamma(v_t - x_t) \rangle + \frac{L}{2} \|\gamma(v_t - x_t)\|^2 \\ &\leq F(x_t) + \gamma \langle \nabla F(x_t), v_t - x_t \rangle + \frac{LD^2\gamma^2}{2}. \end{aligned} \tag{6.3}$$

The first inequality follows since F is L-smooth (see Lemma 6.5.1). The equality is due to the fact that $x_{t+1} - x_t = \gamma(v_t - x_t)$. The second inequality holds because $v_t, x_t \in \Omega$ and because the diameter of Ω is D.

Next, we introduce the following quantity:

$$\hat{v}_t := \arg\max_{v \in \Omega} \langle v, -\nabla F(x_t) \rangle, \tag{6.4}$$

which is used purely for our analysis and is *not* part of the algorithm. For brevity, we use ∇_t to denote $\frac{1}{b} \sum_{i=1}^{b} f(x_t, z_i^t)$.

Rewriting inequality (6.3) using this quantity, we see that

$$\begin{aligned}F(x_{t+1}) &\leq F(x_t) + \gamma\langle \nabla_t, v_t - x_t\rangle + \gamma\langle \nabla F(x_t) - \nabla_t, v_t - x_t\rangle + \frac{LD^2\gamma^2}{2} \\ &\leq F(x_t) + \gamma\langle \nabla_t, \hat{v}_t - x_t\rangle + \gamma\langle \nabla F(x_t) - \nabla_t, v_t - x_t\rangle + \frac{LD^2\gamma^2}{2} \\ &= F(x_t) + \gamma\langle \nabla F(x_t), \hat{v}_t - x_t\rangle + \gamma\langle \nabla F(x_t) - \nabla_t, v_t - \hat{v}_t\rangle + \frac{LD^2\gamma^2}{2} \\ &= F(x_t) - \gamma\mathcal{G}(x_t) + \gamma\langle \nabla F(x_t) - \nabla_t, v_t - \hat{v}_t\rangle + \frac{LD^2\gamma^2}{2} \\ &\leq F(x_t) - \gamma\mathcal{G}(x_t) + D\gamma\|\nabla F(x_t) - \nabla_t\| + \frac{LD^2\gamma^2}{2}. \qquad (6.5)\end{aligned}$$

The second inequality follows from the optimality of v_t in Algorithm 14, while the third inequality follows from recalling that $\mathcal{G}(x_t) = \max_{v\in\Omega}\langle v - x_t, -\nabla F(x_t)\rangle = \langle \hat{v}_t - x_t, -\nabla F(x_t)\rangle$, which holds due to the optimality of $\hat{v}_t$ in (6.4). The last inequality follows from Cauchy-Schwarz and the fact that the diameter of the feasible set Ω is bounded by D.

Taking expectations and using Lemma 6.5.2 in (6.5) we obtain the following important bound:

$$\mathbb{E}[F(x_{t+1})] \leq \mathbb{E}[F(x_t)] - \gamma\mathbb{E}[\mathcal{G}(x_t)] + \frac{GD\gamma}{\sqrt{b}} + \frac{LD^2\gamma^2}{2}.$$

Summing over t and telescoping, we then obtain the upper-bound

$$\begin{aligned}\gamma\sum_{t=0}^{T-1}\mathbb{E}[\mathcal{G}(x_t)] &\leq F(x_0) - \mathbb{E}[F(x_T)] + \frac{TGD\gamma}{\sqrt{b}} + \frac{TLD^2\gamma^2}{2} \\ &\leq F(x_0) - F(x^*) + \frac{TGD\gamma}{\sqrt{b}} + \frac{TLD^2\gamma^2}{2}.\end{aligned}$$

The latter inequality follows from the optimality of x^*. Using the definition of the output x_a of Algorithm 14 and the parameters specified in the theorem statement, we get

$$\begin{aligned}\mathbb{E}[\mathcal{G}(x_a)] &\leq \frac{F(x_0) - F(x^*)}{T\gamma} + \frac{GD}{\sqrt{b}} + \frac{LD^2\gamma}{2} \\ &\leq \frac{D}{\sqrt{T}}\Big(G + \sqrt{\tfrac{2L(F(x_0)-F(x^*))}{\beta}}(1+\beta)\Big), \qquad (6.6)\end{aligned}$$

which concludes the proof of the theorem. □

An immediate consequence of Theorem 6.3.1 is the following complexity result for SFW.

Corollary 6.3.1.1. *Under the setting of Theorem 6.3.1, the SFO complexity and LO complexity of Algorithm 14 are $O(1/\epsilon^4)$ and $O(1/\epsilon^2)$, respectively.*

Proof. The proof follows upon observing that $O(1/\epsilon^2)$ minibatch size is required at each iteration of the algorithm, and noting that as per Theorem 6.3.1 $O(1/\epsilon^2)$ iterations are required to achieve an ϵ-accurate solution. □

Note that the SFO and LO complexity of nonconvex SFW is similar to that of online FW [55] and slightly worse than complexity of SFW for convex problems [56]. Furthermore, for simplicity of analysis, we used a fixed step size and minibatch size. One can derive an essentially similar result using a decreasing step size and increasing minibatch size.

It is important to emphasize that the above results also apply to the finite-sum setting. In particular, when the distribution $\mathcal{P}$ is the empirical measure, then the convergence result in Theorem 6.3.1 also provides convergence rates for the finite-sum case. However, as we will see shortly, these convergence rates can be improved significantly by using variance reduction techniques.

6.3.2 Finite-sum Setting

In this section, we consider the finite-sum setting of (6.1). We show that by building on ideas from variance reduction for SGD, one can significantly improve the convergence rates. The key idea is to use a variance reduced approximation of the gradient [33, 71]. We analyze two different algorithms for the finite-sum setting. Our first algorithm (SVFW) is based on the convex method of [56] adapted to the nonconvex case. Our second algorithm (SAGAFW) is based on another variance reduction technique called SAGA [33].

SVFW Algorithm

Pseudocode of our first method (SVFW) is presented in Algorithm 15. Similar to [71] and [56], nonconvex SVFW is also epoch-based. At the end of each epoch, the full gradient is computed at the current iterate. This gradient is used for controlling the variance of the stochastic gradients in the inner loop. For epoch size $m = 1$, SVFW reduces to the classic FW algorithm. In general, the epoch size m is chosen such that the total number of IFO calls per epoch is $\Theta(n)$. This ensures that the cost of computing the full gradient at the end of each epoch is amortized. To enable a fair comparison with SFW, we assume that the total number of inner iterations across all epochs in Algorithm 15 is T.

We prove the following key result for Algorithm 15. For ease of exposition, we assume that the total number of inner iterations T is a multiple of m.

Theorem 6.3.2. *Consider the finite-sum setting of* (6.1) *where the functions* $\{f_i\}_{i=1}^n$ *are L-smooth. Then, the output* x_a *of Algorithm 15 with parameters* $\gamma_t = \gamma = \sqrt{\frac{F(x_0)-F(x^*)}{TLD^2\beta}}$ *and*

Algorithm 15: SVFW $\left(x_0, T, m, \{\gamma_i\}_{i=0}^{m-1}, \{b_i\}_{i=0}^{m-1}\right)$

```
1: Input: $x_m^0 = x_0 \in \Omega$, epoch size $m$, number of epochs $S = \lceil T/m \rceil$, $\{\gamma_i\}_{i=0}^{m-1}$ where
   $\gamma_i \in [0,1]$ for all $i \in \{0, \dots, m-1\}$, minibatch size $\{b_i\}_{i=0}^{m-1}$
2: for $s = 0$ to $S-1$ do
3:   Let $\tilde{x}^s = x_m^s$
4:   Compute $\tilde{g}^s = \nabla F(\tilde{x}^s) = \frac{1}{n}\sum_{i=1}^n f(\tilde{x}^s)$
5:   for $t = 0$ to $m-1$ do
6:     Uniformly randomly (replacement) select subset $I_t = \{i_1, \dots, i_{b_t}\}$ from $[n]$.
7:     Compute $v_t^{s+1} = \arg\max_{v \in \Omega} \langle v, -\frac{1}{b_t}(\sum_{i \in I_t} \nabla f_i(x_t^{s+1}) - f_i(\tilde{x}^s) + \tilde{g}^s) \rangle$
8:     Compute update direction $d_t^{s+1} = v_t^{s+1} - x_t^{s+1}$
9:     $x_{t+1}^{s+1} = x_t^{s+1} + \gamma_t d_t^{s+1}$
10:  end for
11: end for
12: Output: Iterate $x_a$ chosen uniformly random from $\{\{x_t^{s+1}\}_{t=0}^{m-1}\}_{s=0}^{S-1}$.
```

$b_t = b = m^2$ for all $t \in \{0, \dots, m-1\}$, satisfies

$$\mathbb{E}[\mathcal{G}(x_a)] \leq \frac{2D}{\sqrt{T\beta}} \sqrt{L(F(x_0) - F(x^*))}(1+\beta),$$

where x^* is an optimal solution of (6.1) *and x_a is the output of Algorithm 16.*

Proof. We first analyze the convergence properties of iterates within an epoch. Suppose that the current epoch is $s+1$. For brevity, we drop the symbol s from x_t^{s+1}, $\tilde{x}^s$ and $\tilde{g}^s$, whenever it safe to do so given the context. The first part of the proof is similar to that of Theorem 6.3.1. We use the quantity $\hat{v}_t = \arg\max_{v \in \Omega} \langle v, -\nabla F(x_t) \rangle$, as before, purely for our analysis. For the t^{th} iteration within the epoch s, we have

$$\begin{aligned} F(x_{t+1}) &\leq F(x_t) + \langle \nabla F(x_t), \gamma(v_t - x_t) \rangle + \frac{L}{2}\|\gamma(v_t - x_t)\|^2 \\ &\leq F(x_t) + \gamma\langle \nabla F(x_t), v_t - x_t \rangle + \frac{LD^2\gamma^2}{2}. \end{aligned} \tag{6.7}$$

This is due to Lemma 6.5.1 and definition of x_{t+1} in Algorithm 15. For brevity, we use $\tilde{\nabla}_t$ to denote $\frac{1}{b_t}(\sum_{i \in I_t} \nabla f_i(x_t) - f_i(\tilde{x}) + \tilde{g})$. Rewriting, we then obtain

$$\begin{aligned} F(x_{t+1}) &\leq F(x_t) + \gamma\langle \tilde{\nabla}_t, v_t - x_t \rangle + \gamma\langle \nabla F(x_t) - \tilde{\nabla}_t, v_t - x_t \rangle + \frac{LD^2\gamma^2}{2} \\ &\leq F(x_t) + \gamma\langle \tilde{\nabla}_t, \hat{v}_t - x_t \rangle + \gamma\langle \nabla F(x_t) - \tilde{\nabla}_t, v_t - x_t \rangle + \frac{LD^2\gamma^2}{2} \\ &\leq F(x_t) + \gamma\langle \nabla F(x_t), \hat{v}_t - x_t \rangle + \gamma\langle \nabla F(x_t) - \tilde{\nabla}_t, v_t - \hat{v}_t \rangle + \frac{LD^2\gamma^2}{2} \\ &\leq F(x_t) - \gamma\mathcal{G}(x_t) + D\gamma\|\nabla F(x_t) - \tilde{\nabla}_t\| + \frac{LD^2\gamma^2}{2}. \end{aligned} \tag{6.8}$$

The second inequality is due to the optimality of v_t in Algorithm 15. The last inequality is due to the definition of $\mathcal{G}(x_t)$, the diameter of set Ω, and an application of Cauchy-Schwarz inequality. Note that the above inequality is similar to (6.5), except for the crucial difference in the term $\nabla F(x_t) - \tilde{\nabla}_t$ (instead of $\nabla F(x_t) - \nabla_t$ in (6.5)). As we shall see shortly, this term has much lower variance, which ultimately leads to faster convergence rates.

Taking expectations and using Lemma 6.5.3 in inequality (6.8) we obtain the bound

$$\mathbb{E}[F(x_{t+1})] \leq \mathbb{E}[F(x_t)] - \gamma \mathbb{E}[\mathcal{G}(x_t)] + \frac{LD\gamma}{\sqrt{b}} \mathbb{E}[\|x_t - \tilde{x}\|] + \frac{LD^2\gamma^2}{2}. \tag{6.9}$$

To aid further analysis, we introduce the following Lyapunov function:

$$R_t = \mathbb{E}[F(x_t) + c_t \|x_t - \tilde{x}\|],$$

where $c_m = 0$ and $c_t = c_{t+1} + (LD\gamma)/\sqrt{b}$ for all $t \in \{0, \dots, m-1\}$. Using the relationship in (6.9), we see that

$$\begin{aligned}
R_{t+1} &= \mathbb{E}[F(x_{t+1}) + c_{t+1}\|x_{t+1} - \tilde{x}\|] \\
&\leq \mathbb{E}[F(x_t)] - \gamma \mathbb{E}[\mathcal{G}(x_t)] + \frac{LD\gamma}{\sqrt{b}} \mathbb{E}[\|x_t - \tilde{x}\|] + \frac{LD^2\gamma^2}{2} + c_{t+1}\mathbb{E}[\|x_{t+1} - \tilde{x}\|] \\
&\leq \mathbb{E}[F(x_t)] - \gamma \mathbb{E}[\mathcal{G}(x_t)] + \frac{LD\gamma}{\sqrt{b}} \mathbb{E}[\|x_t - \tilde{x}\|] + \frac{LD^2\gamma^2}{2} + c_{t+1}\mathbb{E}[\|x_{t+1} - x_t\| + \|x_t - \tilde{x}\|] \\
&\leq R_t - \gamma \mathbb{E}[\mathcal{G}(x_t)] + \frac{LD^2\gamma^2}{2} + c_{t+1} D\gamma. \qquad (6.10)
\end{aligned}$$

The second inequality follows from the triangle inequality, while the last inequality holds because: (a) $c_t = c_{t+1} + (LD\gamma)/\sqrt{b}$, and (b) $\|x_{t+1} - x_t\| = \gamma\|v_t - x_t\| \leq D\gamma$ (recall the definition of diameter of Ω). Telescoping over all the iterations within an epoch, we obtain

$$\begin{aligned}
R_m &\leq R_0 - \gamma \sum_{t=0}^{m-1} \mathbb{E}[\mathcal{G}(x_t)] + \frac{LmD^2\gamma^2}{2} + D\gamma \sum_{t=1}^{m} c_t \\
&= R_0 - \gamma \sum_{t=0}^{m-1} \mathbb{E}[\mathcal{G}(x_t)] + \frac{LmD^2\gamma^2}{2} + \frac{L(m-1)mD^2\gamma^2}{2\sqrt{b}}. \qquad (6.11)
\end{aligned}$$

The equality follows from the relationship $c_t = c_{t+1} + (LD\gamma)/\sqrt{b}$. Since $c_m = 0$ and $x_0^{s+1} = \tilde{x}^s = x_m^s$ (in Algorithm 15), from (6.11) we obtain

$$\mathbb{E}[F(x_m^{s+1})] \leq \mathbb{E}[F(x_m^{s+1})] - \gamma \sum_{t=0}^{m-1} \mathbb{E}[\mathcal{G}(x_t^{s+1})] + \frac{LmD^2\gamma^2}{2} + \frac{L(m-1)mD^2\gamma^2}{2\sqrt{b}}.$$

Now telescoping over all epochs, we obtain

$$\mathbb{E}[F(x_m^S)] \leq F(x_0) - \gamma \sum_{s=0}^{S-1} \sum_{t=0}^{m-1} \mathbb{E}[\mathcal{G}(x_t^{s+1})] + \frac{TLD^2\gamma^2}{2} + \frac{TL(m-1)D^2\gamma^2}{2\sqrt{b}}.$$

Rearranging this inequality and using the definition of the output in Algorithm 15, we finally obtain

$$\begin{aligned}\mathbb{E}[\mathcal{G}(x_a)] &\leq \frac{F(x_0) - \mathbb{E}[F(x_m^S)]}{T\gamma} + \frac{LD^2\gamma}{2} + \frac{L(m-1)D^2\gamma}{2\sqrt{b}} \\ &\leq \frac{F(x_0) - F(x^*)}{T\gamma} + LD^2\gamma \\ &\leq 2\sqrt{\frac{LD^2(F(x_0) - F(x^*))}{T\beta}}(1+\beta).\end{aligned}$$

The second inequality follows from the optimality of x^* and because $b = m^2$. The last inequality follows from the choice of γ stated in the theorem. This concludes the proof. □

The analysis suggests that the value of m should be set appropriately in Theorem 6.3.2 to obtain good convergence rates. If m is small, the IFO complexity of Algorithm 15 is dominated by the step involving calculation of the full gradient at the end of each epoch. On the other hand, if m is large, a large minibatch is used in each step of the algorithm (since $b = m^2$), which increases the IFO complexity. With this intuition, we present following important corollary.

Corollary 6.3.2.1. *Under the setting of Theorem 6.3.2 and with $m = \lceil n^{1/3} \rceil$, the IFO complexity and LO complexity of Algorithm 14 are $O(n + n^{2/3}/\epsilon^2)$ and $O(1/\epsilon^2)$, respectively.*

Proof. We first observe that the total number of IFO calls for an epoch (including those required for calculating the full gradient) is $\Theta(m^3 + n)$. Since $m = \lceil n^{1/3} \rceil$, the total amortized IFO complexity of one iteration within an epoch is $O(m^2) = O(n^{2/3})$. Therefore, the IFO complexity is $O(n + n^{2/3}/\epsilon^2)$. Further, since each inner iteration requires $O(1)$ LO calls, the LO complexity is $O(1/\epsilon^2)$. □

SAGAFW Algorithm

SVFW is a semi-stochastic algorithm since it requires calculation of the full gradient at the end of each epoch. Below we propose a purely incremental method (SAGAFW) based on the SAGA algorithm of [33]. The pseudocode for SAGAFW is presented in Algorithm 16.

A key feature of SAGAFW is that it entirely avoids calculation of full gradients. Instead, it updates the average gradient vector g_t at each iteration. This update requires maintaining additional vectors α^i ($i \in [n]$), and in the worst case such a strategy incurs additional storage cost of $O(nd)$. However, this cost can be reduced to $O(n)$ in several practical cases (refer to [33, 141]).

For SAGAFW, we prove the following key result.

```
Algorithm 16: SAGAFW $\left(x_0, T, \{\gamma_i\}_{i=0}^{T-1}, \{b_i\}_{i=0}^{T-1}\right)$
1: Input: $\alpha_0^i = x_0 \in \Omega$ for all $i \in [n]$, number of iterations $T$, $\{\gamma_i\}_{i=0}^{T-1}$ where $\gamma_i \in [0,1]$
   for all $i \in \{0, \dots, T-1\}$, minibatch size $\{b_i\}_{i=0}^{T-1}$
2: Compute $g_0 = \frac{1}{n}\sum_{i=1}^n \nabla F(\alpha_0^i)$
3: for $t = 0$ to $T-1$ do
4:    Uniformly randomly (with replacement) select subsets $I_t, J_t$ from $[n]$ of size $b_t$.
5:    Compute $v_t$ = $\arg\max_{v\in\Omega}\langle v, -\frac{1}{b_t}(\sum_{i\in I_t}\nabla f_i(x_t) - f_i(\alpha_t^i) + g_t)\rangle$
6:    Compute update direction $d_t = v_t - x_t$
7:    $x_{t+1} = x_t + \gamma_t d_t$
8:    $\alpha_{t+1}^j = x_t$ for $j \in J_t$ and $\alpha_{t+1}^j = \alpha_t^j$ for $j \notin J_t$
9:    $g_{t+1} = g_t - \frac{1}{n}\sum_{j\in J_t}(\nabla f_j(\alpha_t^j) - \nabla f_j(\alpha_{t+1}^j))$
10: end for
11: Output: Iterate $x_a$ chosen uniformly random from $\{x_t\}_{t=0}^{T-1}$.
```

Theorem 6.3.3. *Consider the finite-sum setting of* (6.1) *where functions $\{f_i\}_{i=1}^n$ are L-smooth. Define $\theta(b,n,T) = 1/2 + (2n^{3/2}/Tb^{3/2})$. Then the output x_a of Algorithm 16 with parameters $\gamma_t = \gamma = \sqrt{\frac{F(x_0)-F(x^*)}{TLD^2\theta(b,n,T)\beta}}$ and $b_t = b \le n$ for all $t \in \{0,\dots,T-1\}$, satisfies the following:*

$$\mathbb{E}[\mathcal{G}(x_a)] \le \frac{2D}{\sqrt{T\beta}}\sqrt{L\theta(b,n,T)(F(x_0)-F(x^*))}(1+\beta),$$

where x^* is an optimal solution of problem (6.1) *and x_a is the output of Algorithm 16.*

Proof. We use the following quantities in our analysis:

$$\check{\nabla}_t = \frac{1}{b_t}\sum_{i\in I_t}\left(\nabla f_i(x_t) - f_i(\alpha_t^i) + g_t\right)$$
$$\hat{v}_t = \arg\max_{v\in\Omega}\langle v, -\nabla F(x_t)\rangle.$$

The first part of our proof is similar to that of Theorem 6.3.2. Using essentially the same argument until (6.8), we have

$$\begin{aligned}\mathbb{E}[F(x_{t+1})] &\le F(x_t) - \gamma\mathcal{G}(x_t) + D\gamma\|\nabla F(x_t) - \check{\nabla}_t\| + \frac{LD^2\gamma^2}{2} \\ &\le F(x_t) - \gamma\mathcal{G}(x_t) + \frac{LD\gamma\sqrt{n}}{\sqrt{b}}\frac{1}{n}\sum_{i=1}^n\mathbb{E}\|x_t - \alpha_t^i\| + \frac{LD^2\gamma^2}{2}.\end{aligned} \tag{6.12}$$

The second inequality is due to Lemma 6.5.4. Next, we define the following Lyapunov function:

$$R_t = \mathbb{E}[F(x_t)] + c_t\frac{1}{n}\sum_{i=1}^n\mathbb{E}\|x_t - \alpha_t^i\|,$$

where $c_T = 0$ and $c_t = (1-\rho)c_{t+1} + (LD\gamma\sqrt{n})/\sqrt{b}$ for all $t \in \{0, \ldots, T-1\}$, where ρ is the probability $1-(1-1/n)^b$ of an index i being in J_t. We can bound ρ from below as

$$\rho = 1 - \left(1 - \tfrac{1}{n}\right)^b \geq 1 - \tfrac{1}{1+(b/n)} = \tfrac{b/n}{1+b/n} \geq \tfrac{b}{2n}, \tag{6.13}$$

where the first inequality follows from $(1-y)^r \leq 1/(1+ry)$ (which holds for $y \in [0,1]$ and $r \geq 1$), while the second inequality holds because $b \leq n$. Now observe the following:

$$\begin{aligned}
\frac{1}{n}\sum_{i=1}^{n} \mathbb{E}\|x_{t+1} - \alpha_{t+1}^i\| &= \frac{1}{n}\sum_{i=1}^{n} \mathbb{E}\left[\rho\|x_{t+1} - x_t\| + (1-\rho)\|x_{t+1} - \alpha_t^i\|\right] \\
&\leq \frac{1}{n}\sum_{i=1}^{n} \mathbb{E}\Big[\rho\|x_{t+1} - x_t\| + (1-\rho)(\|x_{t+1} - x_t\| + \|x_t - \alpha_t^i\|)\Big] \\
&= \frac{1}{n}\sum_{i=1}^{n} \mathbb{E}\left[\|x_{t+1} - x_t\| + (1-\rho)\mathbb{E}\|x_t - \alpha_t^i\|\right] \qquad (6.14)
\end{aligned}$$

The first equality follows from the definition of α_{t+1}^i in Algorithm 16, while the inequality is just the triangle inequality. Using the above relationship and the bound in (6.12), we obtain

$$\begin{aligned}
R_{t+1} \leq\; & \mathbb{E}[F(x_t)] - \gamma\mathbb{E}[\mathcal{G}(x_t)] + \frac{LD^2\gamma^2}{2} \\
& + \frac{LD\gamma\sqrt{n}}{\sqrt{b}}\frac{1}{n}\sum_{i=1}^{n}\mathbb{E}[\|x_t - \alpha_t^i\|] + c_{t+1}\mathbb{E}[\|x_{t+1} - x_t\|] \\
& + c_{t+1}(1-\rho)\frac{1}{n}\sum_{i=1}^{n}\mathbb{E}[\|x_t - \alpha_t^i\|] \\
\leq\; & R_t - \gamma\mathbb{E}[\mathcal{G}(x_t)] + \frac{LD^2\gamma^2}{2} + c_{t+1}D\gamma. \qquad (6.15)
\end{aligned}$$

The second inequality holds because: (a) $c_t = (1-\rho)c_{t+1} + (LD\gamma\sqrt{n})/\sqrt{b}$, and (b) $\|x_{t+1} - x_t\| = \gamma\|v_t - x_t\| \leq D\gamma$ (due to our bound on the diameter of the set Ω). Telescoping over all the iterations, we see that

$$\begin{aligned}
R_T &\leq R_0 - \gamma\sum_{t=0}^{T-1}\mathbb{E}[\mathcal{G}(x_t)] + \frac{TLD^2\gamma^2}{2} + D\gamma\sum_{t=1}^{T}c_t \\
&\leq R_0 - \gamma\sum_{t=0}^{T-1}\mathbb{E}[\mathcal{G}(x_t)] + \frac{TLD^2\gamma^2}{2} + \frac{LD^2\gamma^2\sqrt{n}}{\rho\sqrt{b}} \\
&\leq R_0 - \gamma\sum_{t=0}^{T-1}\mathbb{E}[\mathcal{G}(x_t)] + \frac{TLD^2\gamma^2}{2} + \frac{2LD^2\gamma^2 n^{3/2}}{b^{3/2}}.
\end{aligned}$$

The second inequality follows form the fact that $\sum_{t=1}^{T} c_t \leq LD\gamma\sqrt{n}/(\rho\sqrt{b})$. This can, in turn, be obtained from the recursion $c_t = (1-\rho)c_{t+1} + (LD\gamma\sqrt{n})/\sqrt{b}$ and $c_T = 0$. The

```
**SVFW-S:**$(x_0, T, B, \gamma)$
Randomly sample $z_1, \cdots, z_B \sim \mathcal{P}$
Let finite-sum $\hat{F}(x) = \frac{1}{B}\sum_{i=1}^{B} f(x, z_i)$
**Output**: SVFW$(x_0, T, B^{1/3}, \gamma, \lceil B^{2/3} \rceil)$
applied on the function $\hat{F}$
```

```
**SAGAFW-S:**$(x_0, T, B, \gamma)$
Randomly sample $z_1, \cdots, z_B \sim \mathcal{P}$
Let finite-sum $\hat{F}(x) = \frac{1}{B}\sum_{i=1}^{B} f(x, z_i)$
**Output**: SAGAFW$(x_0, T, \gamma, \lceil 3B^{1/3} \rceil)$
applied on the function $\hat{F}$
```

Figure 6.2: SVFW-S and SAGAFW-S variants for the stochastic setting.

third inequality is due to the bound on ρ in (6.13). Rearranging the above inequality and using the definition of x_a from Algorithm 16, we finally obtain the bound

$$\begin{aligned}\mathbb{E}[\mathcal{G}(x_a)] &\leq \frac{F(x_0) - \mathbb{E}[F(x_T)]}{T\gamma} + \frac{LD^2\gamma}{2} + \frac{2LD^2\gamma n^{3/2}}{Tb^{3/2}} \\ &\leq \frac{F(x_0) - F(x^*)}{T\gamma} + LD^2\gamma\theta(b, n, T).\end{aligned}$$

The first inequality uses the fact that $c_T = 0$ and $\alpha_0^i = x_0$ (in Algorithm 16). The second inequality uses the optimality of x^* and the definition of $\theta(b, n, T)$. Using the setting of γ in the theorem statement, we obtain the desired result. □

Corollary 6.3.3.1. *Assume $T \geq n$. Under the settings of Theorem 6.3.3 and with $b = \lceil n^{1/3} \rceil$, the IFO and LO complexity of Algorithm 14 are $O(n + n^{1/3}/\epsilon^2)$ and $O(1/\epsilon^2)$, respectively.*

Proof. First, observe that for $T \geq n$ and $b = \lceil n^{1/3} \rceil$, $\theta(b, n, T) \leq 5/2$ in Theorem 6.3.3. Thus, the IFO complexity is $O(n + n^{1/3}/\epsilon^2)$. Furthermore, since each iteration requires just $O(1)$ LO calls, the LO complexity is $O(1/\epsilon^2)$. □

Notably, the IFO complexity of SAGAFW is lower than that of SVFW. Moreover, if $T \geq n^{3/2}$ and $b = 1$, then we have $\theta(b, n, T) \leq 5/2$, in which case the IFO complexity is $O(n^{3/2} + 1/\epsilon^2)$.

6.4 Variance Reduction in Stochastic Setting

In this section, we improve the convergence rates in the stochastic setting using variance reduction techniques. The key idea is to first obtain samples $\{z_i\}$ are chosen independently according to the distribution $\mathcal{P}$ and then use SVFW or SAGAFW, described in this chapter, on the finite-sum problem over these samples. The pseudocode for the SVFW and SAGAFW variants for stochastic setting (SVFW-S and SAGAFW-S respectively) are provided in Figure 6.2. The following is the key result regarding the convergence rates of SVFW-S and SAGAFW-S.

Theorem 6.4.1. *Consider the stochastic setting of* (6.1) *where f is G-Lipschitz and $f(., z)$ is L-smooth for all $z \in \Xi$. Suppose $B = T$ and $\gamma = \sqrt{\frac{F(x_0) - F(x^*)}{TLD^2\beta}}$ (for* SVFW*-S and* SAGAFW*-S).*

Then the output of SVFW-*S and* SAGAFW-*S satisfy the following:*

$$\mathbb{E}[\mathcal{G}(x_a)] \leq \frac{2D}{\sqrt{T\beta}}\sqrt{L(F(x_0) - F(x^*))}(1+\beta) + \frac{GD}{\sqrt{T}} \tag{6.16}$$

Proof. Consider the finite-sum $\hat{F}(x) = \frac{1}{B}\sum_{i=1}^{B} f(x, z_i)$ where $z_1, \cdots, z_B \sim \mathcal{P}$. We use the following notation:

$$\hat{\mathcal{G}}(x) = \max_{v \in \Omega} \langle v - x, -\nabla \hat{F}(x) \rangle.$$

Let $\bar{v}_t^{s+1} = \arg\max_{v\in\Omega}\langle v - x_t^{s+1}, -\nabla F(x_t^{s+1})\rangle$ and $\hat{v}_t^{s+1} = \arg\max_{v\in\Omega}\langle v - x_t^{s+1}, -\nabla \hat{F}(x_t^{s+1})\rangle$. We first observe the following key relationship for SVFW:

$$\begin{aligned}
\mathbb{E}[\mathcal{G}(x_t^{s+1}) - \hat{\mathcal{G}}(x_t^{s+1})] &= \mathbb{E}[\langle \bar{v}_t^{s+1} - x_t^{s+1}, -\nabla F(x_t^{s+1})\rangle] - \mathbb{E}[\langle \hat{v}_t^{s+1} - x_t^{s+1}, -\nabla \hat{F}(x_t^{s+1})\rangle] \\
&\leq \mathbb{E}[\langle \bar{v}_t^{s+1} - x_t^{s+1}, -\nabla F(x_t^{s+1})\rangle] - \mathbb{E}[\langle \bar{v}_t^{s+1} - x_t^{s+1}, -\nabla \hat{F}(x_t^{s+1})\rangle] \\
&\leq \mathbb{E}[\langle \bar{v}_t^{s+1} - x_t^{s+1}, \nabla \hat{F}(x_t^{s+1}) - \nabla F(x_t^{s+1})\rangle] \\
&\leq D\mathbb{E}[\|\nabla \hat{F}(x_t^{s+1}) - \nabla F(x_t^{s+1})\|] \leq \frac{GD}{\sqrt{T}}.
\end{aligned}$$

The first inequality is due to the optimality of $\hat{v}_t^{s+1}$. The third inequality follows from Cauchy-Schwarz inequality. The last inequality is due to Lemma 6.5.2. Adding the above inequality across all the iterations and epochs, we get:

$$\mathbb{E}[\mathcal{G}(x_a)] \leq \mathbb{E}[\hat{\mathcal{G}}(x_a)] + \frac{GD}{\sqrt{T}}.$$

Using the bound on $\mathbb{E}[\hat{\mathcal{G}}(x_a)]$ in Theorem 6.3.2 (here, recall we are using SVFW on $\hat{F}$) in the above inequality, we get the desired result. The proof for SAGAFW-S is similar. □

The following corollary on the complexity of SVFW-S and SAGAFW-S is immediate consequence of the above result.

Corollary 6.4.1.1. *Under the setting of Theorem 6.4.1, the SFO complexity of* SVFW-*S and* SAGAFW-*S (in Figure 6.2) are $O(1/\epsilon^{10/3})$ and $O(1/\epsilon^{8/3})$, respectively. The LO complexity of both* SVFW-*S and* SAGAFW-*S is $O(1/\epsilon^2)$.*

Proof. The proof follows from the fact that $B = T$, $b = \lceil B^{2/3} \rceil$ (in SVFW-S) and $b = \lceil 3B^{1/3} \rceil$ (in SAGAFW-S) and IFO complexities of SVFW and SAGAFW given in Corollary 6.3.2.1 and 6.3.3.1 respectively. □

By comparing Corollary 6.4.1.1 with Corollary 6.3.1.1, we see that SVFW-S and SAGAFW-S have better SFO complexity than SFW.

6.5 Discussion

It is important to remark on the complexity results derived in this chapter. For the stochastic setting, we showed that the SFO and LO complexity of SFW are $O(1/\epsilon^4)$ and $O(1/\epsilon^2)$, respectively. At first glance, these rates might appear worse than those obtained for nonconvex SGD (see [47]). However, it is important to note that the convergence criterion used in this chapter is different from the one used in [47]. It is an important piece of future work to understand the precise relationship between these convergence criteria. Furthermore, the convergence rates in this chapter are similar to those obtained for online Frank-Wolfe [55] and only slightly worse than those obtained for stochastic Frank-Wolfe in the *convex setting* [56]. We, further, improved the convergence rate of SFW by using variance reduction ideas in the stochastic setting (SVFW-S and SAGAFW-S algorithms in Section 6.4). Understanding the tightness of these rates is an interesting open problem left as future work.

For the finite-sum setting, while the complexity results of SFW still hold, we obtained significantly faster convergence rates by using variance reduction techniques. The dependence of IFO and LO complexity of nonconvex SVFW and SAGAFW, on ϵ is $O(1/\epsilon^2)$, which matches the classical Frank-Wolfe algorithm [78]. However, SVFW and SAGAFW exhibit a much weaker dependence on n than FW; wherein, they are provably faster than the classical Frank-Wolfe by a factor of $n^{1/3}$ and $n^{2/3}$, respectively. Similar (but not same) benefits have also been reported for nonconvex SVRG and SAGA over gradient descent [139, 141]. Interestingly, there appears to be a gap between the convergence rates of SVFW and SAGAFW. Whether this gap is an artifact of our analysis or has deeper reasons remains open.

We conclude with a remark on a subtle point regarding the step size γ. The step size γ in Theorems 6.3.1, 6.3.2, and 16 requires knowledge of parameters like L, D and $F(x) - F(x^*)$. Typically, an estimate of the these values suffices in practice. In absence of such knowledge, one can completely eliminate this dependence of γ on these parameters by simply choosing $\beta = \frac{2(F(x_0)-F(x^*))}{LD^2}$. Fortunately, this comes at the cost of only slightly worse *constants* in the convergence rate.

Appendix: Omitted Proofs

The following bound on the value of functions with Lipschitz continuous gradients is classical (see e.g., [111]).

Lemma 6.5.1. *If $f : \mathbb{R}^d \to \mathbb{R}$ is L-smooth, then*

$$f(x) \leq f(y) + \langle \nabla f(y), x - y \rangle + \frac{L}{2}\|x - y\|^2,$$

for all $x, y \in \mathbb{R}^d$.

The following lemma is useful for bounding the variance of the gradient estimate used in the stochastic setting.

Lemma 6.5.2. *Suppose the function $F(x) = \mathbb{E}_z[f(x,z)]$ where z is a random variable with distribution $\mathcal{P}$ and support Ξ, and $\max_{z\in\Xi} \|\nabla f(x,z)\| \leq G$ for all $x \in \Omega$. Also, let $\bar{\nabla}_x = \frac{1}{b}\sum_{i\in I_t} \nabla f(x,z_i)$ where $\{z_i\}_{i=1}^b$ are i.i.d. samples from the distribution $\mathcal{P}$. Then, the following holds for any $x \in \Omega$:*

$$\mathbb{E}[\|\bar{\nabla}_x - \nabla F(x)\|] \leq \frac{G}{\sqrt{b}}.$$

Proof. The proof follows from a simple application of Lemma 6.5.5 and Jensen's inequality. □

The following result is useful for bounding the variance of the updates of SVFW and follows from a slight modification of a result in [139]. We give the proof here for completeness.

Lemma 6.5.3 ([139]). *Let $\tilde{\nabla}_t = \frac{1}{b_t}(\sum_{i\in I_t} \nabla f_i(x_t^{s+1}) - f_i(\tilde{x}^s) + \tilde{g}^s)$ in Algorithm 15. For the iterates x_t^{s+1} and $\tilde{x}^s$ where $t \in \{0,\dots,m-1\}$ and $s \in \{0,\dots,S-1\}$ in Algorithm 15, the following inequality holds:*

$$\mathbb{E}_{I_t}[\|\nabla F(x_t^{s+1}) - \tilde{\nabla}_t\|] \leq \frac{L}{\sqrt{b_t}}\|x_t^{s+1} - \tilde{x}^s\|.$$

Proof. For the ease of exposition, we first define

$$\zeta_t^{s+1} = \frac{1}{|I_t|}\sum_{i\in I_t}\left(\nabla f_i(x_t^{s+1}) - \nabla f_i(\tilde{x}^s)\right).$$

Using this notation, we then obtain the following:

$$\begin{aligned}
&\mathbb{E}_{I_t}[\|\nabla F(x_t^{s+1}) - \tilde{\nabla}_t\|^2] \\
&= \mathbb{E}_{I_t}[\|\zeta_t^{s+1} + \nabla F(\tilde{x}^s) - \nabla F(x_t^{s+1})\|^2] \\
&= \mathbb{E}_{I_t}[\|\zeta_t^{s+1} - \mathbb{E}_{I_t}[\zeta_t^{s+1}]\|^2] \\
&= \frac{1}{b_t^2}\mathbb{E}_{I_t}\left[\left\|\sum_{i\in I_t}\left(\nabla f_i(x_t^{s+1}) - \nabla f_i(\tilde{x}^s) - \mathbb{E}_{I_t}[\zeta_t^{s+1}]\right)\right\|^2\right].
\end{aligned}$$

The second equality is due to the fact that $\mathbb{E}_{I_t}[\zeta_t^{s+1}] = \nabla F(x_t^{s+1}) - \nabla F(\tilde{x}^s)$. From the above relationship, we get

$$\begin{aligned}
&\mathbb{E}_{I_t}[\|\nabla F(x_t^{s+1}) - \tilde{\nabla}_t\|^2] \\
&\leq \frac{1}{b_t}\mathbb{E}_{I_t}\left[\sum_{i\in I_t}\|\nabla f_i(x_t^{s+1}) - \nabla f_i(\tilde{x}^s) - \mathbb{E}_{I_t}[\zeta_t^{s+1}]\|^2\right] \\
&\leq \frac{1}{b_t}\mathbb{E}_{I_t}\left[\sum_{i\in I_t}\|\nabla f_i(x_t^{s+1}) - \nabla f_i(\tilde{x}^s)\|^2\right] \\
&\leq \frac{L^2}{b_t}\|x_t^{s+1} - \tilde{x}^s\|^2.
\end{aligned}$$

The first inequality follows from Lemma 6.5.5. The second inequality is due to the fact that for a random variable ζ, $\mathbb{E}[\|\zeta - \mathbb{E}[\zeta]\|^2] \leq \mathbb{E}[\|\zeta\|^2]$. The last inequality follows from L-smoothness of f_i. The result follows from a simple application of Jensen's inequality to the inequality above. □

The following result is important for bounding the variance in SAGAFW. The key difference from Lemma 6.5.3 is that the variance term in SAGAFW involves α_t^i. Again, we provide the proof for completeness.

Lemma 6.5.4. *Let $\check{\nabla}_t = \frac{1}{b_t}(\sum_{i \in I_t} \nabla f_i(x_t) - f_i(\alpha_t^i) + g_t)$ in Algorithm 16. For the iterates x_t, v_t and $\{\alpha_t^i\}_{i=1}^n$ where $t \in \{0, \dots, T-1\}$ in Algorithm 16, we have the inequality*

$$\mathbb{E}_{I_t}[\|\nabla F(x_t) - \check{\nabla}_t\|] \leq \frac{L}{\sqrt{b_t}} \sum_{i=1}^n \frac{1}{\sqrt{n}} \|x_t - \alpha_t^i\|.$$

Proof. As before we first define the quantity

$$\zeta_t = \frac{1}{|I_t|} \sum_{i \in I_t} \left(\nabla f_i(x_t) - \nabla f_i(\alpha_t^i) \right).$$

With this notation, we then obtain the equality

$$\begin{aligned}
&\mathbb{E}_{I_t}[\|\nabla F(x_t) - \check{\nabla}_t\|^2] \\
&= \mathbb{E}_{I_t}\left[\left\|\zeta_t + \frac{1}{n}\sum_{i=1}^n \nabla f_i(\alpha_t^i) - \nabla F(x_t)\right\|^2\right] = \mathbb{E}_{I_t}[\|\zeta_t - \mathbb{E}_{I_t}[\zeta_t]\|^2] \\
&= \frac{1}{b^2}\mathbb{E}_{I_t}\left[\left\|\sum_{i \in I_t}\left(\nabla f_i(x_t) - \nabla f_i(\alpha_t^i) - \mathbb{E}_{I_t}[\zeta_t]\right)\right\|^2\right].
\end{aligned}$$

The second equality follows from the fact that $\mathbb{E}_{I_t}[\zeta_t] = \nabla F(x_t) - \frac{1}{n}\sum_{i=1}^n \nabla f_i(\alpha_t^i)$. From the above inequality, we get the following bound:

$$\begin{aligned}
&\mathbb{E}_{I_t}[\|\nabla F(x_t) - \check{\nabla}_t\|^2] \\
&\leq \frac{1}{b_t}\mathbb{E}_{I_t}\left[\sum_{i \in I_t} \|\nabla f_i(x_t) - \nabla f_i(\alpha_t^i) - \mathbb{E}_{I_t}[\zeta_t]\|^2\right] \\
&\leq \frac{1}{b_t}\mathbb{E}_{I_t}\left[\sum_{i \in I_t} \|\nabla f_i(x_t) - \nabla f_i(\alpha_t^i)\|^2\right] \leq \frac{L^2}{nb_t}\sum_{i=1}^n \|x_t - \alpha_t^i\|^2.
\end{aligned}$$

The first inequality is due to Lemma 6.5.5, while the second inequality holds because for a random variable ζ, $\mathbb{E}[\|\zeta - \mathbb{E}[\zeta]\|^2] \leq \mathbb{E}[\|\zeta\|^2]$. The last inequality is from L-smoothness of f_i ($i \in [n]$) and uniform randomness of the set I_t. By applying Jensen's inequality, we get the desired result. □

Lemma 6.5.5. *For random variables $z_1, \dots, z_r$ that are independent and have mean 0, we have*

$$\mathbb{E}\left[\|z_1 + ... + z_r\|^2\right] = \mathbb{E}\left[\|z_1\|^2 + ... + \|z_r\|^2\right].$$

Proof. Expanding the left hand side we have

$$\mathbb{E}\left[\|z_1 + ... + z_r\|^2\right] = \sum_{i,j=1}^{r} \mathbb{E}\left[z_i z_j\right] = \mathbb{E}\left[\textstyle\sum_{i=1}^{r} \|z_i\|^2\right];$$

the second equality here follows from the our hypothesis. □

Chapter 7

Variance Reduced Stochastic Langevin Dynamics

7.1 Introduction

In the previous chapters, we examined the benefit of variance reduction in the context of optimization. Here, we demonstrate that these techniques can be useful in settings beyond optimization by investigating its benefit for Bayesian posterior Inference (BI). BI involves drawing samples from a posterior distribution based on the training data and is a central task in machine learning and Bayesian analysis. It has widespread applications in science, law, sports and philosophy. While sampling methods like Monte Carlo are considered as gold standard in Bayesian posterior inference due to their asymptotic convergence properties, their convergence can be slow in large models due to poor mixing. Gradient-based Monte Carlo methods such as Langevin Dynamics and Hamiltonian Monte Carlo [104] allow us to use gradient information to more efficiently explore posterior distributions over continuous-valued parameters. By traversing contours of a potential energy function based on the posterior distribution, these methods allow us to make large moves in the sample space. Although gradient-based methods are efficient in exploring the posterior distribution, they are limited by the computational cost of computing the gradient and evaluating the likelihood on large datasets. As a result, stochastic variants are a popular choice when working with large data sets [172].

Stochastic gradient methods [149] have long been used in the optimization community to decrease the computational cost of gradient-based optimization algorithms such as gradient descent. These methods replace the (expensive, but accurate) gradient evaluation with a noisy (but computationally cheap) gradient evaluation on a random subset of the data. With appropriate scaling, this gradient evaluated on a random subset of the data acts as a proxy for the true gradient. A carefully designed schedule of step sizes ensures convergence of the stochastic algorithm.

A similar idea has been employed to design stochastic versions of gradient-based Monte Carlo methods [7, 8, 97, 172]. By evaluating the derivative of the log likelihood on only a small subset of data points, we can drastically reduce computational costs.

However, using stochastic gradients comes at a cost: While the resulting estimates are unbiased, they do have very high variance. This leads to an increased probability of selecting paths with high deviation from the true gradient, leading to slower convergence.

There have been a number of variations proposed on the basic stochastic gradient Langevin dynamics (SGLD) model of [172]: [26] incorporate a momentum term to improve posterior exploration; [37] propose using additional variables to stabilize fluctuations; [119] proposed modifications to facilitate exploration of simplex; [49] provides sampling solutions for correlated data. However, none of these methods directly tries to reduce the variance in the computed stochastic gradient.

As was the case with the original SGLD algorithm, we look to the optimization community for inspiration, since high variance is also detrimental in stochastic gradient based optimization. A plethora of variance reduction techniques have been recently proposed to alleviate this issue for the stochastic gradient descent (SGD) algorithm [33, 71, 153]. By incorporating a carefully designed (usually unbiased) term into the update sequence of SGD, these methods reduce the variance that arises due to the stochastic gradients in SGD; thereby, providing strong theoretical and empirical performance.

Inspired by these successes in the optimization community, we propose methods for reducing the variance in stochastic Langevin dynamics. Our approach bridges the gap between the faster (in terms of iterations) convergence of non-stochastic Langevin dynamics, and the faster per-iteration speed of stochastic Langevin dynamics. While our approach draws its motivation from stochastic optimization literature, it is to our knowledge the first approach that aims to directly reduce variance in a gradient-based Monte Carlo method. While our focus is on Langevin dynamics, our approach is easily applicable to other gradient-based Monte Carlo methods.

Main Contributions: We propose a new Langevin algorithm designed to reduce variance in the stochastic gradient, with minimal additional computational overhead. We also provide a memory efficient variant of our algorithm. We demonstrate theoretical conversion to the true posterior under reasonable assumptions, and show that the rate of convergence has a tighter bound than one previously shown for stochastic Langevin dynamics. We complement these theoretical results with empirical evaluation showing impressive speed-ups versus a standard stochastic Langevin algorithm, on a variety of machine learning tasks such as regression, classification, independent component analysis and mixture modeling.

7.2 Preliminaries

Let $\mathbf{X} = \{x_i\}_{i=1}^N$ be a set of data items modeled using a likelihood function $p(X|\theta) = \prod_{i=1}^N p(x_i|\theta)$ where the parameter θ has prior distribution $p(\theta)$. We are interested in sampling from the posterior distribution $p(\theta|\mathbf{X}) \propto p(\theta) \prod_{i=1}^N p(x_i|\theta)$. If N is large, standard Langevin Dynamics is not feasible due to the high cost of repeated gradient evaluations; a more scalable approach is to use a stochastic variant [172] (which we will refer to as stochastic gradient Langevin dynamics, or SGLD). SGLD uses a classical Robbins-Monro stochastic approximation to the true gradient [149]. At each iteration t of the algorithm, a subset of the data $\mathbf{X}_t = \{x_{t1}, \ldots, x_{tn}\}$ is sampled and the parameters are updated by using only this subset of data, according to

$$\Delta\theta_t = \frac{h_t}{2}\left(\nabla \log p(\theta_t) + \frac{N}{n}\sum_{i=1}^{n} \nabla \log p(x_{ti}|\theta_t)\right) + \eta_t \tag{7.1}$$

where $\eta_t \sim N(0, h_t)$, and h_t is the learning rate. h_t is set in such a fashion that $\sum_{t=1}^{\infty} h_t = \infty$ and $\sum_{t=1}^{\infty} h_t^2 < \infty$. This provides an approximation to a first order Langevin diffusion, with dynamics

$$d\theta = -\tfrac{1}{2}\nabla_\theta U dt + dW, \tag{7.2}$$

where U is the unnormalized negative log posterior. Equation (7.2) has stationary distribution $\rho(\theta) \propto \exp\{-U(\theta)\}$. Let $\bar{\phi} = \int \phi(\theta)\rho(\theta)d\theta$ where ϕ represents a test function of interest. For a numerical method that generates samples $\{\theta_t\}_{i=0}^{T-1}$, the empirical average $\frac{1}{T}\sum_{t=0}^{T-1}\phi(\theta_t)$ is denoted by $\hat{\phi}$. Furthermore, let ψ denote the solution to the Poisson equation $\mathcal{L}\psi = \phi - \bar{\phi}$, where $\mathcal{L}$ is the generator of the diffusion, given by

$$\mathcal{L}\psi = \langle \nabla_\theta \psi, \nabla_\theta U \rangle + \tfrac{1}{2}\sum_i \nabla_i^2 \psi. \tag{7.3}$$

The decreasing step size h_t in our approximation (Equation (7.1)) means we do not have to incorporate a Metropolis-Hastings step to correct for the discretization error relative to Equation (7.2); however it comes at the cost of slowing the mixing rate of the algorithm. We note that, while the discretized Langevin diffusion is Markovian, its convergence guarantees rely on the quality of the approximation, rather than from standard Markov chain Monte Carlo analyses.

A second source of error comes from the use of stochastic approximations to the true gradients. This is equivalent to using an approximate generator $\tilde{\mathcal{L}}_t = \mathcal{L} + \Delta V_t$ where $\Delta V_t = \nabla_\theta \psi \cdot (\nabla_\theta U_t - \nabla_\theta U)$ where U_t is the current stochastic approximation to U. Our key contribution in this chapter will be replacing the Robbins-Monroe approximation to U with a lower-variance approximation, thus reducing the error.

To see more clearly the effect of the variance of our stochastic approximation on the estimator error, we present a result derived for SGLD by [25]:

Theorem 7.2.1. *[25] Let U_t be an unbiased estimate of U and $h_t = h$ for all $t \in \{1, \ldots, T\}$. Then under certain reasonable assumptions (concretely, assumption [A1] in Section 7.4), for a smooth test function ϕ, the MSE of* SGLD *at time $K = hT$ is bounded, for some $C > 0$ independent of (T, h) in the following manner:*

$$\mathbb{E}(\hat{\phi} - \bar{\phi})^2 \leq C\left(\underbrace{\frac{\frac{1}{T}\sum_t \mathbb{E}[\|\Delta V_t\|^2]}{T}}_{T_1} + \frac{1}{Th} + h^2\right). \tag{7.4}$$

Here $\|.\|$ represents the operator norm.

We clearly see that the MSE depends on the variance term $\mathbb{E}[\|\Delta V_t\|^2]$, which in turn depends on the variance of the noisy stochastic gradients. Since, for consistency, we require $h \to 0$ as $T \to \infty$,[1] provided $\mathbb{E}[\|\Delta V_t\|^2]$ is bounded by a constant τ, the term T_1 ceases to dominate as $T \to \infty$, meaning that the effect of noise in the stochastic gradient becomes negligible. However outside this asymptotic regime, the effect of the variance term in Equation (7.4) remains significant. This motivates our efforts in this chapter to decrease the variance of the approximate gradient, while maintaining an unbiased estimator.

One easy approach to decrease the variance is by using larger minibatches. However, this comes at a considerably large computational cost; thus, undermining the whole benefit of using SGLD. Inspired by the recent success of variance reduction techniques in stochastic optimization [33, 71, 153], we take a rather different approach to reduce the effect of noisy gradients.

7.3 Variance Reduction for Langevin Dynamics

As we have seen in Section 7.2, reducing the variance of our stochastic approximation can reduce our estimation error. In this section, we introduce two approaches for variance reduction, based on recent variance reduction algorithms for gradient descent [33, 71]. The first algorithm, SAGA-LD, is appropriate when our bottleneck is computation; it yields improved convergence with minimal additional computational costs over SGLD. The second algorithm, SVRG-LD, is appropriate when our bottleneck is memory; while the computational cost is generally higher than SAGA-LD, the memory requirement is lower, with the memory overhead beyond that of stochastic Langevin dynamics scales as $O(d)$. In practice, we found that computation was a greater bottleneck in the examples considered, so our experimental section only focuses on SAGA-LD; however on larger datasets with easily computable gradients, SVRG-LD may be the optimal choice.

7.3.1 SAGA-LD

The increased variance in SGLD is due to the fact that we only have information from $n \ll N$ data points at each iteration. However, inspired by a minibatch version of the SAGA algorithm [33], we can include information from the remaining data points via an approximate gradient and partially update the average gradient in each operation. We call this approach SAGA-LD.

Under SAGA-LD, we explicitly store N approximate gradients $\{g_{\alpha i}\}_{i=1}^N$, corresponding to the N data points. Concretely, let $\alpha_t = (\alpha_t^i)_{i=1}^N$ be a set of vectors, initialized as $\alpha_0^i = \theta_0$ for all $i \in [N]$, and initialize $g_{\alpha i} = \nabla \log p(x_i|\alpha_0^i)$. As we iterate through the data, if a data point is not selected in the current minibatch, we approximate its gradient with $g_{\alpha i}$. If $I_t = \{i_{1t}, \ldots i_{nt}\}$ is the minibatch selected at iteration t, this means we

[1]In particular, if $h \propto T^{-1/3}$, we obtain the optimal convergence rate for the above upper bound.

Algorithm 17: SAGA-LD

```
1: Input: $\alpha_0^i = \theta_0 \in \mathbb{R}^d$ for $i \in \{1, \dots, N\}$, step sizes $\{h_t > 0\}_{i=0}^{T-1}$
2: $g_\alpha = \sum_{i=1}^N \nabla \log p(x_i|\alpha_0^i)$
3: for $t = 0$ to $T-1$ do
4:     Uniformly randomly pick a set $I_t$ from $\{1, \dots, N\}$ (with replacement) such that
       $|I_t| = b$
5:     Randomly draw $\eta_t \sim N(0, h_t)$
6:     $\theta_{t+1} = \theta_t + \frac{h_t}{2}\left(\nabla \log p(\theta_t) + \frac{N}{n}\sum_{i \in I_t}(\nabla \log p(x_i|\theta_t) - \nabla \log p(x_i|\alpha_t^i)) + g_\alpha\right) + \eta_t$
7:     $\alpha_{t+1}^i = \theta_t$ for $i \in I_t$ and $\alpha_{t+1}^i = \alpha_t^i$ for $i \notin I_t$
8:     $g_\alpha = g_\alpha + \sum_{i \in I_t}(\nabla \log p(x_i|\alpha_{t+1}^i) - \nabla \log p(x_i|\alpha_t^i))$
9: end for
10: Output: Iterates $\{\theta_t\}_{t=0}^{T-1}$.
```

approximate the gradient as

$$\sum_{i=1}^N \nabla \log p(x_i|\theta_t) \approx \frac{N}{n}\sum_{i \in I_t}\left(\nabla \log p(x_i|\theta_t) - g_{\alpha i}\right) + g_\alpha \tag{7.5}$$

When Equation (7.5) is used for MAP estimation it corresponds to SAGA[33]. However by injecting noise into the parameter update in the following manner

$$\Delta\theta_t = \frac{h_t}{2}\left(\nabla \log p(\theta_t) + \frac{N}{n}\sum_{i \in I_t}\left(\nabla \log p(x_i|\theta_t) - g_{\alpha i}\right) + g_\alpha\right) + \eta_t, \text{ where } \eta_t \sim N(0, h_t) \tag{7.6}$$

we can adapt it for sampling from the posterior. After updating $\theta_{t+1} = \theta_t + \Delta\theta_t$, we let $\alpha_{t+1}^i = \theta_t$ for $i \in I_t$. Note that we do not need to explicitly store the α_t^i; instead we just update the corresponding gradients $g_{\alpha i}$ and average gradient. The SAGA-LD algorithm is summarized in Algorithm 17.

The approximation in Equation (7.6) gives an unbiased estimate of the true gradient, since the minibatch I_t is sampled uniformly at random from $[N]$, and the α_i^t are independent of I_t. SAGA-LD offers two key properties: (i) As shown in Section 7.4, SAGA-LD has a better convergence properties than SGLD (ii) The computational overhead is minimal, since SAGA-LD does not require calculation of the full gradient after every few iterations. Instead, it simply makes use of gradients that are already being calculated in the current minibatch. Combined, we end up with a similar computational complexity to SGLD, with a much better convergence rate.

The only downside of SAGA-LD, when compared with SGLD, is in terms of memory storage. Since we need to store N individual gradients $g_{\alpha i}$, we typically have a storage overhead of $O(Nd)$ relative to SGLD. Fortunately, in many applications of interest to machine learning, the cost can be reduced to $O(N)$ (please refer to [33] for more details), and in practice the cost of the higher memory requirements is outweighed by the improved convergence and low computational cost.

Algorithm 18: SVRG-LD

```
1: Input: $\tilde{\theta} = \theta_0 \in \mathbb{R}^d$, epoch length $m$, step sizes $\{h_t > 0\}_{i=0}^{T-1}$
2: for $t = 0$ to $T - 1$ do
3:     if ($t$ mod $m$ = 0) then
4:         $\tilde{\theta} = \theta_t$
5:         $\tilde{g} = \sum_{i=1}^{N} \nabla \log p(x_i|\tilde{\theta})$
6:     end if
7:     Uniformly randomly (with replacement) pick a set $I_t$ from $\{1, \dots, N\}$ such that
       $|I_t| = n$
8:     Randomly draw $\eta_t \sim N(0, h_t)$
9:     $\theta_{t+1} = \theta_t + \frac{h_t}{2}\left(\nabla \log p(\theta_t) + \frac{N}{n}\sum_{i \in I_t}(\nabla \log p(x_i|\theta_t) - \nabla \log p(x_i|\tilde{\theta})) + \tilde{g}\right) + \eta_t$
10: end for
11: Output: Iterates $\{\theta_t\}_{t=0}^{T-1}$.
```

7.3.2 SVRG-LD

If the memory overhead of SAGA-LD is not acceptable, we can use a variant that reduces storage requirements, at the cost of higher computational demands. The memory complexity for SAGA-LD is high because the average gradient $\tilde{g}$ is updated at each step. This can be avoided by updating the average gradient at every m iterations in a single evaluation, and never storing the individual gradients $g_{\alpha i}$. Concretely, after every m passes through the data, we evaluate the gradient on the entire data set, to obtain an estimate $\tilde{g} = \sum_{i=1}^{N} \tilde{g}_i$, where $\tilde{g}_i = \nabla \log p(x_i|\tilde{\theta})$ is the local gradient evaluated at that time point. This yields an update of the form

$$\Delta\theta_t = \frac{h_t}{2}\left(\nabla \log p(\theta_t) + \frac{N}{n}\sum_{i \in I_t}\left(\nabla \log p(x_i|\theta_t) - \tilde{g}_i\right) + \tilde{g}\right) + \eta_t \text{ where } \eta_t \sim N(0, h_t) \tag{7.7}$$

Without adding noise η_t the update sequence in Equation (7.7) corresponds to the stochastic variance reduction gradient descent algorithm [71]. Pseudocode for this procedure is given in Algorithm 18.

While the memory requirements are lower, the computational cost is higher, due to the cost of a full update of $\tilde{g}$. Further, convergence may be negatively effected due to the fact that, as we move further from $\tilde{\theta}$, $\tilde{g}$ will be further from the true gradient. In practice, we found SAGA-LD to be a more effective algorithm on the datasets considered. We relegate further details about SVRG-LD to the appendix.

7.4 Analysis

Our motivation in this chapter was to improve the convergence of SGLD, by reducing the variance of the gradient estimate. As we saw in Theorem 7.2.1, a high variance $\mathbb{E}[||\Delta V_t||^2]$, corresponding to noisy stochastic gradients, leads to a large bound on the

MSE of a test function. We expand this analysis to show that the algorithms introduced in this chapter exhibit a tighter bound.

Theorem 7.2.1 required a number of assumptions, given below in [A1]. Discussion of the reasonableness of these assumptions is provided in [25].

[A1] We assume the functional ψ that solves the Poisson equation $\mathcal{L}\psi = \phi - \bar{\phi}$ is bounded up to 3rd-order derivatives by some function Γ, i.e., $\|\mathcal{D}^k\psi\| \le C_k\Gamma^{p_k}$ where $\mathcal{D}$ is the kth order derivative (for $k = (0, 1, 2, 3)$), and C_k, $p_k > 0$. We also assume that the expectation of Γ on $\{\theta_t\}$ is bounded ($\sup_t \mathbb{E}\Gamma^p[\theta_t] < \infty$) and that Γ is smooth such that $\sup_{s\in(0,1)} \Gamma^p(s\theta + (1-s)\theta') \le C(\Gamma^p(\theta) + \Gamma^p(\theta'))$, $\forall \theta, \theta'$, $p \le \max 2p_k$ for some $C > 0$.

In our analysis of SAGA-LD and SVRG-LD, we make the assumptions in A1, and add the following further assumptions about the smoothness of our gradients:

[A2] We assume that the functions $\log p(x_i|\theta)$ are Lipschitz smooth with constant L for all , $i \in [N]$ i.e., $\|\nabla \log p(x_i|\theta) - \nabla \log p(x_i|\theta')\| \le L\|\theta - \theta'\|$ for all $i \in [N]$ and $\theta, \theta' \in \mathbb{R}^d$. We assume that $(\Delta V_t \psi(\theta))^2 \le C'\|U_t(\theta) - U(\theta)\|^2$ for some constant $C' > 0$ for all $\theta \in \mathbb{R}^d$, where ψ is the solution to the Poisson equation for our test function. We also assume that $\|\nabla \log p(\theta)\| \le \sigma$ and $\|\nabla \log p(x_i|\theta)\| \le \sigma$ for some σ and all $i \in [N]$ and $\theta \in \mathbb{R}^d$.

The Lipschitz smoothness assumption is very common both in the optimization literature [111] and when working with Itô diffusions [25]. The bound on $(\Delta V_t \psi(\theta))^2$ holds when the gradient $\|\nabla\psi\|$ is bounded.

Loosely, these assumptions encode the idea that the gradients don't change too quickly, so that we limit the errors introduced by incorporating gradients based on previous values of θ. With these assumptions, we state the following key results for SAGA-LD and SVRG-LD, which are proved in the appendix.

Theorem 7.4.1. *Let $h_t = h$ for all $t \in \{1, \dots, T\}$. Under the assumptions [A1],[A2], for a smooth test function ϕ, the MSE of* SAGA-LD *(in Algorithm 17) at time $K = hT$ is bounded, for some $C > 0$ independent of (T, h) in the following manner:*

$$\mathbb{E}(\hat{\phi} - \bar{\phi})^2 \le C\left(\frac{N^2 \min\{\sigma^2, \frac{N^2}{n^2}(L^2h^2\sigma^2 + hd)\}}{nT} + \frac{1}{Th} + h^2\right). \tag{7.8}$$

A similar result can be shown for SVRG-LD in Algorithm 18. In particular, we have the following key result for SVRG-LD.

Theorem 7.4.2. *Let $h_t = h$ for all $t \in \{1, \dots, T\}$. Under the assumptions [A1],[A2], for a smooth test function ϕ, the MSE of* SVRG-LD *(in Algorithm 18) at time $K = hT$ is bounded, for some $C > 0$ independent of (T, h) in the following manner:*

$$\mathbb{E}(\hat{\phi} - \bar{\phi})^2 \le C\left(\frac{N^2 \min\{\sigma^2, m^2(L^2h^2\sigma^2 + hd)\}}{nT} + \frac{1}{Th} + h^2\right). \tag{7.9}$$

The result in Theorem 7.4.2 is qualitatively equivalent to that in Theorem 7.4.1 when $m = \lfloor N/n \rfloor$. In general, such a choice of m is preferable because in this case the overall cost of calculation of full gradient in Algorithm 18 becomes insignificant.

In order to assess the theoretical convergence of our proposed algorithm, we compare the bounds for SVRG-LD (Theorem 7.4.2) and SAGA-LD (Theorem 7.4.1) with those obtained for SGLD (Theorem 7.2.1. Under the assumptions in this section, it is easy to show that the term T_1 in Theorem 7.2.1 becomes $O(N^2\sigma^2/(Tn))$. In contrast, both Theorem 7.4.1 and 7.4.2 show that due to a reduction in variance, SVRG-LD and SAGA-LD exhibit a much weaker dependence. More specifically, this is manifested in the form of the following bound:

$$\frac{N^2 \min\{\sigma^2, \frac{N^2}{n^2}(h^2\sigma^2 + hd)\}}{nT}.$$

Note that this is tighter than the corresponding bound on SGLD. We also note that, similar to SGLD, SAGA-LD and SVRG-LD require $h \to 0$ as $T \to \infty$. In such a scenario, the convergence becomes significantly faster relative to SGLD as $h \to 0$.

7.5 Experiments

We present our empirical results in this section. We focus on applying our stochastic gradient method to four different machine learning tasks, carried out on benchmark datasets: (i) Bayesian linear regression (ii) Bayesian logistic regression and (iii) Independent component analysis (iv) Mixture models. We focus on SAGA-LD, since in the applications considered, the convergence and computational benefits of SAGA-LD are more beneficial than the memory benefits of SVRG-LD;

In order to reduce the initial computational costs associated with calculating the initial average gradient, we use a variant of Algorithm 17 that calculates g_α (in Algorithm 7.4.1) in an online fashion and reweights the updates accordingly. Note that such a heuristic is also commonly used in the implementation of SAG and SAGA in the context of optimization [33, 153].

In all our experiments, we use a decreasing step size for SGLD as suggested by [172]. In particular, we use $\epsilon_t = a(b+t)^{-\gamma}$, where the parameters a, b and γ are chosen for each dataset to give the best performance of the algorithm on that particular dataset. For SAGA-LD, due to the benefit of variance reduction, we use a simple two phase constant step size selection strategy. In each of these phases, a constant step size is chosen such that SAGA-LD gives the best performance on the particular dataset. The minibatch size, n, in both SGLD and SAGA-LD is held at a constant value of 10 throughout our experiments. All algorithms are initialized to the same point and the same sequence of minibatches is pre-generated and used in both algorithms.

7.5.1 Regression

We first demonstrate the performance of our algorithm on Bayesian regression. Formally, we are provided with inputs $\mathbf{Z} = \{x_i, y_i\}_{i=1}^N$ where $x_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$. The distribution of the i^{th} output y_i is given by $p(y_i|x_i) = \mathcal{N}(\beta^\top x_i, \sigma_e)$, where $p(\beta) = \mathcal{N}(0, \lambda^{-1}I)$. Due to conjugacy, the posterior distribution over β is also normal, and the gradients of

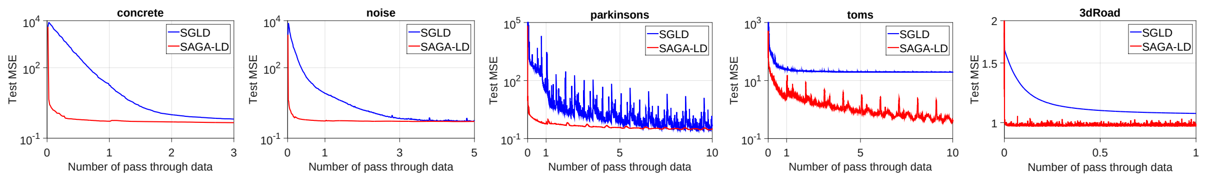


Figure 7.1: Performance comparison of SGLD and SAGA-LD on a regression task. The x-axis and y-axis represent the number of passes through the entire data and the average test MSE, respectively. Additional experiments are provided in the appendix.

the log-likelihood and the log-prior are given by $\nabla_\beta \log(P(y_i|x_i, \beta)) = -(y_i - \beta^T x_i)x_i$ and $\nabla_\beta \log(P(\beta)) = -\lambda\beta$. We ran experiments on 11 standard UCI regression datasets, summarized in Table 7.1.[2] In each case, we set the prior precision $\lambda = 1$, and we partitioned our dataset into training (70%), validation (10%) and test (20%) sets. The validation set is used to select the step size parameters, and we report the mean square error (MSE) evaluated on the test set, using 5-fold cross-validation.

The average test MSE on a subset of datasets is reported in Figure 7.1. We relegate the remaining experiments to the Appendix. As shown in Figure 7.1, SAGA-LD converges much faster than the SGLD method (taking less than one pass through the whole dataset in many cases). This performance gain is consistent across all the datasets. Furthermore, the step size selection was much simpler for SAGA-LD than SGLD.

Datasets	concrete	noise	parkinson	bike	toms	protein	casp	kegg	3droad	music	twitter
N	1030	1503	5875	17379	45730	45730	53500	64608	434874	515345	583250
P	8	5	21	12	96	9	9	27	2	90	77

Table 7.1: Summary of datasets used in regression.

7.5.2 Classification

We next turn our attention to the classification task, using Bayesian logistic regression. In this case, the input is the set $\mathbf{Z} = \{x_i, y_i\}_{i=1}^N$ where $x_i \in \mathbb{R}^d$, $y_i \in \{0, 1\}$. The distribution of the output y_i for given sample x_i is given by $y_i = \phi(\beta^T x_i)$, where $p(\beta) = \mathcal{N}(0, \lambda^{-1} I)$ and $\phi(z) = 1/(1 + exp(-z))$. Here, the gradient of the log-likelihood and the log-prior are given by $\nabla_\beta \log(P(y_i|x_i, \beta)) = (y_i - \phi(\beta^T x_i))x_i$ and $\nabla_\beta \log(P(\beta)) = -\lambda\beta$ respectively. Again, the value of λ is set to 1 for all our classification experiments, and the dataset split and the parameter selection method is exactly same as that in our regression experiments. We run experiments on five binary classification datasets in the UCI repository, summarized in Table 7.2, and report the the test log-likelihood for each datasets, using 5-fold cross validation. Figure 7.2 shows the performance of SGLD and SAGA-LD for the classification datasets. As we saw with the regression task, SAGA-LD converges faster that SGLD on all the datasets, thus, demonstrating the efficiency of the our algorithm in this setting.

[2]The datasets can be downloaded from `https://archive.ics.uci.edu/ml/index.html`

Datasets	pima	diabetic	eeg	space	susy
N	768	1151	14980	58000	100000
d	8	20	15	9	18

Table 7.2: Summary of the datasets used for classification.

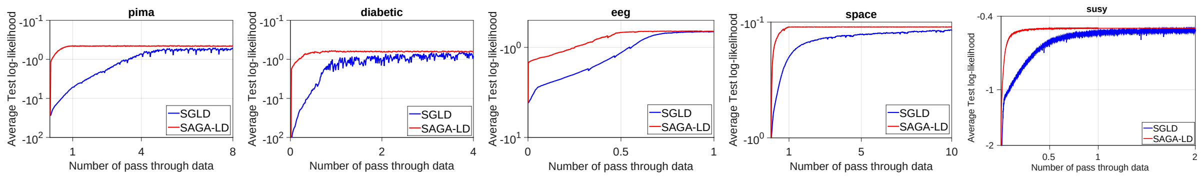


Figure 7.2: Comparison of performance of SGLD and SAGA-LD for Bayesian logistic regression. The x-axis and y-axis represent the number of effective passes through the dataset and the test log-likelihood respectively in these plots.

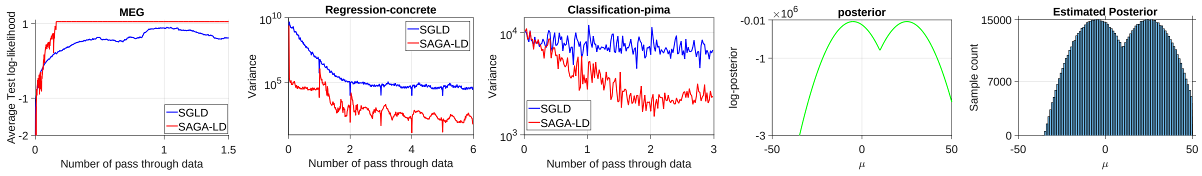


Figure 7.3: Left plot shows the performance of SGLD and SAGA-LD for ICA task. The next two plots show the variance of SGLD and SAGA-LD for regression, classification. The rightmost two plot shows true and estimated posterior using SAGA-LD for Mixture model

7.5.3 Bayesian Independent Component Analysis

Under Bayesian Independent Component Analysis (ICA), we assume that a dataset $\mathbf{x} = \{x_i\}_{i=1}^N$ is distributed according to

$$p(\mathbf{x}|W) \propto |\det(W)| \prod_{i=1}^d p(y_i), \quad W_{ij} \sim \mathcal{N}(0, \lambda), \tag{7.10}$$

where $W \in \mathbb{R}^{d \times d}$, $y_i = w_i^T x$ and $p(y_i) = 1/(4\cosh^2(\frac{1}{2}y_i))$. The gradient of the log-likelihood and the log-prior are $\nabla_W \log(p(x_i|W)) = (W^{-1})^T - Y_i x_i^T$ where $Y_{ij} = \tanh(\frac{1}{2}y_{ij})$ for all $j \in [d]$ and $\nabla_W \log(p(W)) = -\lambda W$ respectively. All other parameters are set as before. We used a standard ICA dataset for our experiment[3] This dataset comprises 17730 time-points with 122 channels from which we extract the first 10 channels. We omit the details of the experimental setup because it is similar to that for regression and classification. The performance (in terms of test set log likelihood) of SGLD and SAGA-LD for the ICA task is shown in Figure 7.3. As seen in Figure 7.3, similar to the regression and classification tasks, SAGA-LD outperforms SGLD in the ICA task.

[3]The dataset can be downloaded from https://www.cis.hut.fi/projects/ica/eegmeg/MEG_data.html.

7.5.4 Mixture Model

Finally we evaluate how well SAGA-LD estimates the true posterior of parameters of mixture models. We generate $\{x_i\}_{i=1}^N$ is generated from a mixture of two Gaussians given by $p(x|\mu, \sigma_1, \sigma_2, \gamma) = \frac{1}{2}\mathcal{N}(x; \mu, \sigma_1^2) + \frac{1}{2}\mathcal{N}(x; -\mu + \gamma, \sigma_2^2)$. We only try to estimate the posterior distribution over μ while the rest is kept fixed. We sample points from $p(x|\mu)$ with $\mu = -5, \gamma = 20$. The 2 plots on the right of Figure 7.3 show that we are able to estimate the true posterior correctly from the samples.

Discussion: Our experiments provide a very compelling reason to use variance reduction techniques for SGLD, complementing the theoretical justification given in Section 7.4. The hypothesized variance reduction is demonstrated in Figure 7.3, where we compare the variances of SGLD and SAGA-LD as compared to true gradient on regression and classification tasks. As we see from all of the experimental results in this section, SAGA-LD converges with relatively very few samples in comparison to SGLD. This is, specially, important in Bayesian averaging models where, typically, the size of the model is proportional to the number of samples from the posterior distribution. Thus, with SAGA-LD, we can achieve better performance with very few samples. Another advantage is that, while we require the step size to tend to zero, we can use a much simpler schedule than SGLD.

7.6 Discussion and Future Work

SAGA-LD is a new stochastic Langevin method that obtains improved convergence by reducing the variance in the stochastic gradient. An alternative method, SVRG-LD, can be used when memory is at a premium. For both SAGA-LD and SVRG-LD, we proved a tighter convergence bound than the one previously shown for stochastic Langevin dynamic. We also showed on a variety of machine learning task SAGA-LD converges to the true posterior faster than SGLD, suggesting the widespread use of SAGA-LD in place of SGLD. We note that, unlike other stochastic Langevin methods, our sampler is non-Markovian. Since our convergence guarantees are based on bounding the error relative to the full Langevin diffusion rather than on properties of a Markov chain, this does not impact the validity of our sampler.

While we showed the efficacy of using our proposed variance reduction technique to SGLD, our proposed strategy is very generic enough and can also be applied to other gradient-based MCMC techniques such as [7, 8, 37, 97, 119]. We leave this as future work.

Appendix: Omitted Proofs and Additional Experiments

We provide details of the theoretical analysis provided in the chapter. We first start with the proof of Theorem 7.4.2 and then look at the proof of Theorem 7.4.1.

7.7 Proof of Theorem 7.4.2

We introduce a notation for simplifying our theoretical exposition. For $t \in [sm, (s+1)m)$ for some integer $s \in [0, \lfloor T/m \rfloor]$, let $\tilde{\theta}_t = \theta^{sm}$. Then the update of SVRG-LD can rewritten in the following manner:

$$\theta_{t+1} = \theta_t + \frac{h_t}{2}\left(\nabla \log p(\theta_t) + \frac{N}{n}\sum_{i \in I_t}(\nabla \log p(x_i|\theta_t) - \nabla \log p(x_i|\tilde{\theta}_t)) + \sum_{i=1}^{N} \nabla \log p(x_i|\tilde{\theta}_t)\right) + \eta_t.$$

We use the a key result proved in [25] for general stochastic gradient MCMC. First, recall that $\bar{\phi} = \frac{1}{T}\sum_t \phi(\theta_t)$ and MSE is $\mathbb{E}(\phi - \bar{\phi})^2$. We have the following important bound for MSE of SGLD:

$$\mathbb{E}(\phi - \bar{\phi})^2 \leq C\left(\frac{\frac{1}{T}\sum_t \mathbb{E}(\Delta V_t \psi(\theta_t))^2}{T} + \frac{1}{Th} + h^2\right). \tag{7.11}$$

for some constant $C > 0$ (refer to [25] for a detailed proof of this fact). Using Assumption [A1], we get the result in Theorem 7.2.1. We use a different upper bound for the term $\frac{1}{T}\sum_t \mathbb{E}(\Delta V_t \psi(\theta_t))^2$. In particular, we have the following upper bound:

$$\begin{aligned}
&\frac{1}{C'T}\sum_t \mathbb{E}(\Delta V_t \psi(\theta_t))^2 \leq \frac{1}{T}\sum_t \mathbb{E}[\|U_t(\theta_t) - U(\theta_t)\|^2] \\
&= \frac{1}{T}\sum_t \mathbb{E}\left\|\frac{N}{n}\sum_{i \in I_t}(\nabla \log p(x_i|\theta_t) - \nabla \log p(x_i|\tilde{\theta}_t)) + \sum_{i=1}^{N}\nabla \log p(x_i|\tilde{\theta}_t) - \sum_{i=1}^{N}\nabla \log p(x_i|\theta_t)\right\|^2 \\
&= \frac{1}{Tn^2}\sum_t \mathbb{E}\Bigg\|\sum_{i \in I_t}\Bigg(N\left[\nabla \log p(x_i|\theta_t) - \nabla \log p(x_i|\tilde{\theta}_t)\right] \\
&\qquad\qquad - \left[\sum_{i=1}^{N}\nabla \log p(x_i|\theta_t) - \sum_{i=1}^{N}\nabla \log p(x_i|\tilde{\theta}_t)\right]\Bigg)\Bigg\|^2 \\
&= \frac{1}{Tn^2}\sum_t \mathbb{E}\sum_{i \in I_t}\Bigg\|\Bigg(N\left[\nabla \log p(x_i|\theta_t) - \nabla \log p(x_i|\tilde{\theta}_t)\right] \\
&\qquad\qquad - \left[\sum_{i=1}^{N}\nabla \log p(x_i|\theta_t) - \sum_{i=1}^{N}\nabla \log p(x_i|\tilde{\theta}_t)\right]\Bigg)\Bigg\|^2 \\
&\leq \frac{1}{Tn^2}\sum_t \mathbb{E}\sum_{i \in I_t}\left\|N\left[\nabla \log p(x_i|\theta_t) - \nabla \log p(x_i|\tilde{\theta}_t)\right]\right\|^2 \leq \frac{L^2N^2}{Tn}\sum_t \mathbb{E}\left\|\theta_t - \tilde{\theta}_t\right\|^2.
\end{aligned} \tag{7.12}$$

The first inequality follows from our assumption [A3] in Section 7.4. The third equality follows from Lemma 5.12.7. The second inequality is due to the fact that $\mathbb{E}[\|\zeta - \mathbb{E}[\zeta]\|^2] \leq \mathbb{E}[\|\zeta\|^2]$ for any random variable $\zeta \in \mathbb{R}^d$. The last inequality is follows from the Lipschitz continuity of $\nabla \log p(x_i|\theta)$. Note that alternatively, we can also bound in the following fashion:

$$\frac{1}{C'T}\sum_t \mathbb{E}(\Delta V_t \psi(\theta_t))^2 \le \frac{1}{Tn^2}\sum_t \mathbb{E}\sum_{i\in I_t}\left\|N\left[\nabla \log p(x_i|\theta_t) - \nabla \log p(x_i|\tilde{\theta}_t)\right]\right\|^2 \le \frac{2N^2\sigma^2}{n} \tag{7.13}$$

Consider $t \in [sm+1,(s+1)m)$ for some integer $s \in [0, \lfloor T/m \rfloor]$. We bound $\mathbb{E}\|\theta_t - \tilde{\theta}\|^2$ in the following manner.

$$\begin{aligned}\mathbb{E}\left\|\theta_t - \tilde{\theta}_t\right\|^2 &= \mathbb{E}\left\|\sum_{j=sm}^{t-1}(\theta_{j+1}-\theta_j)\right\|^2 \\ &\le (t-sm)\sum_{j=sm}^{t-1}\mathbb{E}[\|\theta_{j+1}-\theta_j\|^2] \le m\sum_{j=sm}^{t-1}\mathbb{E}[\|\theta_{j+1}-\theta_j\|^2].\end{aligned} \tag{7.14}$$

The first inequality is due to Lemma 7.11.2. The second inequality is due to the fact that $t \in [sm+1,(s+1)m)$. We bound the term $\mathbb{E}[\|\theta_{j+1}-\theta_j\|^2]$ in the following manner:

$$\begin{aligned}&\mathbb{E}[\|\theta_{j+1}-\theta_j\|^2] \\ &= E\left\|\frac{h}{2}\left(\nabla \log p(\theta_j) + \frac{N}{n}\sum_{i\in I_t}(\nabla \log p(x_i|\theta_j) - \nabla \log p(x_i|\tilde{\theta}_j)) + \sum_{i=1}^N \nabla \log p(x_i|\tilde{\theta}_j)\right) + \eta_j\right\|^2 \\ &\le \frac{3h^2}{4}E\|\nabla \log p(\theta_j)\|^2 + 3\mathbb{E}[\|\eta_j\|^2] \\ &\qquad + \frac{3h^2}{4}\left\|\frac{N}{n}\sum_{i\in I_t}(\nabla \log p(x_i|\theta_j) - \nabla \log p(x_i|\tilde{\theta}_j)) + \sum_{i=1}^N \nabla \log p(x_i|\tilde{\theta}_j)\right\|^2 \\ &\le \frac{3h^2\sigma^2}{4} + 3hd + \frac{3h^2}{4}\left\|\frac{N}{n}\sum_{i\in I_t}(\nabla \log p(x_i|\theta_j) - \nabla \log p(x_i|\tilde{\theta}_j)) + \sum_{i=1}^N \nabla \log p(x_i|\tilde{\theta}_j)\right. \\ &\qquad\qquad \left. - \sum_{i=1}^N \nabla \log p(x_i|\theta_j) + \sum_{i=1}^N \nabla \log p(x_i|\theta_j)\right\|^2 \\ &\le \frac{3h^2\sigma^2}{4} + 3hd + \frac{3h^2}{2}\left\|\sum_{i=1}^N \nabla \log p(x_i|\theta_j)\right\|^2 \\ &\quad + \frac{3h^2}{2}\left\|\frac{N}{n}\sum_{i\in I_t}(\nabla \log p(x_i|\theta_j) - \nabla \log p(x_i|\tilde{\theta}_j)) + \sum_{i=1}^N \nabla \log p(x_i|\tilde{\theta}_j) - \sum_{i=1}^N \nabla \log p(x_i|\theta_j)\right\|^2 \\ &\le \frac{3h^2\sigma^2}{4} + 3hd + \frac{3N^2h^2\sigma^2}{2} + \frac{3N^2h^2\sigma^2}{n}.\end{aligned}$$

The first inequality follows from Lemma 7.11.2 (with $r = 3$). The second inequality follows from the fact that $\|\nabla p(\theta)\|^2 \leq \sigma^2$ for all $\theta \in \mathbb{R}^d$ and the fact that $\eta_j \sim N(0, \sqrt{h}\mathbb{I})$. The third inequality again follows from Lemma 7.11.2 with $r = 2$. The last inequality follows from: (a) Lemma 7.11.2 with $r = N$ and (b) bound in Equation (7.12). Substituting the bound in Equation (7.14), we get the following:

$$\mathbb{E}\left\|\theta_t - \tilde{\theta}_t\right\|^2 \leq m^2\left[\frac{3h^2\sigma^2}{4} + 3hd + \frac{3N^2h^2\sigma^2}{2} + \frac{3N^2h^2\sigma^2}{n}\right]. \tag{7.15}$$

Substituting Equation (7.15) in Equation (7.12) and substituting the minimum of the resultant bound and bound in Equation (7.13) into Equation (7.11) gives the desired result.

7.8 Proof of Theorem 7.4.1

The proof of Theorem 7.4.1 is along the lines of Theorem 7.4.2. But the key difficulty, in comparison to the analysis of SVRG-LD, comes from the fact that the full gradient is not computed after every few epochs. We again start with the following inequality in [25]:

$$\mathbb{E}(\phi - \bar{\phi})^2 \leq C\left(\frac{\frac{1}{T}\sum_t \mathbb{E}(\Delta V_t \psi(\theta_t))^2}{T} + \frac{1}{Th} + h^2\right). \tag{7.16}$$

for some constant $C > 0$. For SAGA-LD, we have the following inequality:

$$\begin{aligned}
&\frac{1}{C'T}\sum_t \mathbb{E}(\Delta V_t \psi(\theta_t))^2 \leq \frac{1}{T}\sum_t \mathbb{E}[\|U_t(\theta_t) - U(\theta_t)\|^2] \\
&= \frac{1}{T}\sum_t \mathbb{E}\left\|\frac{N}{n}\sum_{i\in I_t}(\nabla \log p(x_i|\theta_t) - \nabla \log p(x_i|\alpha_t^i)) + \sum_{i=1}^N \nabla \log p(x_i|\alpha_t^i) - \sum_{i=1}^N \nabla \log p(x_i|\theta_t)\right\|^2 \\
&= \frac{1}{Tn^2}\sum_t \mathbb{E}\left\|\sum_{i\in I_t}\left(N\left[\nabla \log p(x_i|\theta_t) - \nabla \log p(x_i|\alpha_t^i)\right]\right.\right. \\
&\qquad\qquad \left.\left. - \left[\sum_{i=1}^N \nabla \log p(x_i|\theta_t) - \sum_{i=1}^N \nabla \log p(x_i|\alpha_t^i)\right]\right)\right\|^2 \\
&= \frac{1}{Tn^2}\sum_t \mathbb{E}\sum_{i\in I_t}\left\|\left(N\left[\nabla \log p(x_i|\theta_t) - \nabla \log p(x_i|\alpha_t^i)\right]\right.\right. \\
&\qquad\qquad \left.\left. - \left[\sum_{i=1}^N \nabla \log p(x_i|\theta_t) - \sum_{i=1}^N \nabla \log p(x_i|\alpha_t^i)\right]\right)\right\|^2 \\
&\leq \frac{1}{Tn^2}\sum_t \mathbb{E}\sum_{i\in I_t}\left\|N\left[\nabla \log p(x_i|\theta_t) - \nabla \log p(x_i|\alpha_t^i)\right]\right\|^2 \leq \frac{L^2N}{Tn}\sum_t\sum_i \mathbb{E}\left\|\theta_t - \alpha_t^i\right\|^2.
\end{aligned} \tag{7.17}$$

The first inequality is due to our assumption [A3] in Section 7.4. The third equality is obtained by using Lemma 7.11.1. The second inequality is due to the fact that $\mathbb{E}[\|\zeta - \mathbb{E}[\zeta]\|^2] \leq \mathbb{E}[\|\zeta\|^2]$ for any random variable $\zeta \in \mathbb{R}^d$. The last inequality is follows from the Lipschitz continuity of $\nabla \log p(x_i|\theta)$ and uniform randomness of the set I_t.

Let $\gamma = 1 - (1 - 1/N)^n$. γ represents the probability that an index is chosen at a particular iteration. Our goal is to bound the term $\sum_t \sum_i \mathbb{E} \left\|\theta_t - \alpha_t^i\right\|^2$. To this end, we observe the following:

$$\begin{aligned}
\mathbb{E} \left\|\theta_t - \alpha_t^i\right\|^2 &= \sum_{j=0}^{t-1} \mathbb{E} \left[\mathbb{E} \left[\left\|\theta_t - \alpha_t^i\right\|^2 \mid \alpha_t^i = \theta_j\right]\right] \\
&\leq \sum_{j=0}^{t-1} (t-j)^2 \left[\frac{3h^2\sigma^2}{4} + 3hd + \frac{3N^2h^2\sigma^2}{2} + \frac{3N^2h^2\sigma^2}{n}\right] P(\alpha_t^i = \theta_j) \\
&= \left[\frac{3h^2\sigma^2}{4} + 3hd + \frac{3N^2h^2\sigma^2}{2} + \frac{3N^2h^2\sigma^2}{n}\right] \sum_{j=0}^{t-1} (t-j)^2 (1-\gamma)^{t-j-1} \gamma \\
&= \left[\frac{3h^2\sigma^2}{4} + 3hd + \frac{3N^2h^2\sigma^2}{2} + \frac{3N^2h^2\sigma^2}{n}\right] \gamma \sum_{j=1}^{t} j^2 (1-\gamma)^{j-1} \\
&\leq \left[\frac{3h^2\sigma^2}{4} + 3hd + \frac{3N^2h^2\sigma^2}{2} + \frac{3N^2h^2\sigma^2}{n}\right] \gamma \sum_{j=1}^{\infty} j^2 (1-\gamma)^{j-1} \\
&\leq \frac{2}{\gamma^2} \left[\frac{3h^2\sigma^2}{4} + 3hd + \frac{3N^2h^2\sigma^2}{2} + \frac{3N^2h^2\sigma^2}{n}\right] \\
&\leq \frac{8N^2}{n^2} \left[\frac{3h^2\sigma^2}{4} + 3hd + \frac{3N^2h^2\sigma^2}{2} + \frac{3N^2h^2\sigma^2}{n}\right]. \qquad (7.18)
\end{aligned}$$

The first equality is due to law of total expectation. The first inequality is due can be obtained by using similar argument as the one in Equation (7.15). The second inequality follows from simple calculation of $P(\alpha_t^i = x_j)$. This in turn uses the fact that the set I_t is selected uniformly randomly at each iteration. The last equality is due to the standard formula: $\sum_{j=1}^{\infty} j^2(1-\gamma)^{j-1} = (2-\gamma)/\gamma^3$. The last inequality is due to the following bound on γ:

$$\gamma = 1 - \left(1 - \frac{1}{N}\right)^n \geq 1 - \frac{1}{1 + \frac{n}{N}} = \frac{n/N}{1 + n/N} \geq \frac{n}{2N}. \qquad (7.19)$$

The first inequality is due to the fact that $(1-x)^n \leq 1/(1+nx)$ for $x \in [0,1]$ and $n \in N$. The last inequality is due to the fact that $n/N \leq 1$. Substituting the bound in Equation (7.18) into Equation (7.17), we have

$$\frac{1}{C'T} \sum_t \mathbb{E}(\Delta V_t \psi(\theta_t))^2 \leq \frac{L^2N^3}{n^2} \left[\frac{3h^2\sigma^2}{4} + 3hd + \frac{3N^2h^2\sigma^2}{2} + \frac{3N^2h^2\sigma^2}{n}\right]. \qquad (7.20)$$

Algorithm 19: SVRG-LD

```
1: Input: $\tilde{\theta} = \theta_0 \in \mathbb{R}^d$, epoch length $m$, step sizes $\{h_t > 0\}_{i=0}^{T-1}$
2: for $t = 0$ to $T - 1$ do
3:     if ($t$ mod $m$ = 0) then
4:         $\tilde{\theta} = \theta_t$
5:         $\tilde{g} = \sum_{i=1}^{N} \nabla \log p(x_i|\tilde{\theta})$
6:     end if
7:     Uniformly randomly (with replacement) pick a set $I_t$ from $\{1, \dots, N\}$ such that
       $|I_t| = n$
8:     Randomly draw $\eta_t \sim N(0, h_t)$
9:     $\theta_{t+1} = \theta_t + \frac{h_t}{2}\left(\nabla \log p(\theta_t) + \frac{N}{n}\sum_{i \in I_t}(\nabla \log p(x_i|\theta_t) - \nabla \log p(x_i|\tilde{\theta})) + \tilde{g}\right) + \eta_t$
10: end for
11: Output: Iterates $\{\theta_t\}_{t=0}^{T-1}$.
```

Note that similar to Equation (7.13), the following bound holds for SAGA-LD.

$$\frac{1}{C'T}\sum_t \mathbb{E}(\Delta V_t \psi(\theta_t))^2 \le \frac{1}{Tn^2}\sum_t \mathbb{E}\sum_{i \in I_t}\left\|N\left[\nabla \log p(x_i|\theta_t) - \nabla \log p(x_i|\alpha_t^i)\right]\right\|^2 \le \frac{2N^2\sigma^2}{n} \tag{7.21}$$

Using the minimum of the bounds in Equation (7.21) and (7.20) in Equation (7.16) gives us the desired result.

7.9 SVRG-LD

The memory complexity for SAGA-LD is high because the average gradient $\tilde{g}$ is updated at each step. This can be avoided by updating the average gradient at every m iterations in one expensive evaluation. Concretely, every m passes through the data, we evaluate the gradient on the entire data set, to obtain an estimate $\tilde{g} = \sum_{i=1}^{N} \tilde{g}_i$, where $\tilde{g}_i = \nabla \log p(x_i|\tilde{\theta})$ is the local gradient evaluated at every m step.

As we iterate through the data, if a data point is not selected in the current minibatch, we approximate its gradient with $\tilde{g}_i$. If $I_t = \{i_{1t}, \dots i_{nt}\}$ is the minibatch selected at iteration t, then we approximate $\sum_{i=1}^{N} \nabla \log p(x_i|\theta_t)$ our update is

$$\sum_{i=1}^{N} \nabla \log p(x_i|\theta_t) \approx \frac{N}{n}\sum_{i \in I_t}(\nabla \log p(x_i|\theta_t) - \tilde{g}_i) + \tilde{g}, \tag{7.22}$$

This yields an update of the form

$$\delta\theta_t = \frac{h_t}{2}\left(\nabla \log p(\theta_t) + \frac{N}{n}\sum_{i \in I_t}(\nabla \log p(x_i|\theta_t) - \tilde{g}_i) + \tilde{g}\right) + \eta_t \tag{7.23}$$

where $\eta_t \sim N(0, h_t)$. Pseudocode for this procedure is given in Algorithm 18.

As with SAGA-LD, we note that the update in Equation (7.7) gives an unbiased estimate of the true gradient, since I_t is chosen uniformly at random (with replacement) from $[N] = \{1, \dots, N\}$, since we have $\mathbb{E}[\frac{N}{n}\sum_{i \in I_t} \tilde{g}_i - \tilde{g}] = 0$. Therefore, the term $\frac{N}{n}\sum_{i \in I_t} \tilde{g}_i - \tilde{g}$ does not add any biased to the stochastic gradient.

By calculating the full gradient after every m iterations, we ensure that the accuracy of the approximation is not allowed to decrease too significantly, and ensure that the variance of the updates is controlled. We provide concrete bounds in Section 7.4. We note that, if $m \geq \lfloor N/n \rfloor$, the computational complexity of SVRG-LD is similar to SGLD.

One desirable property of SVRG-LD is that it has low memory requirements: SVRG requires just $O(d)$ extra memory (in order to store the average gradient $\tilde{g}$), in comparison with SGLD. However, there is a potentially large computational burden due to the need to periodically calculate the full gradient. In practice, we found that computation was a greater bottleneck in the examples considered, so our experimental section focuses on SAGA-LD.

7.10 Other Experiment Results

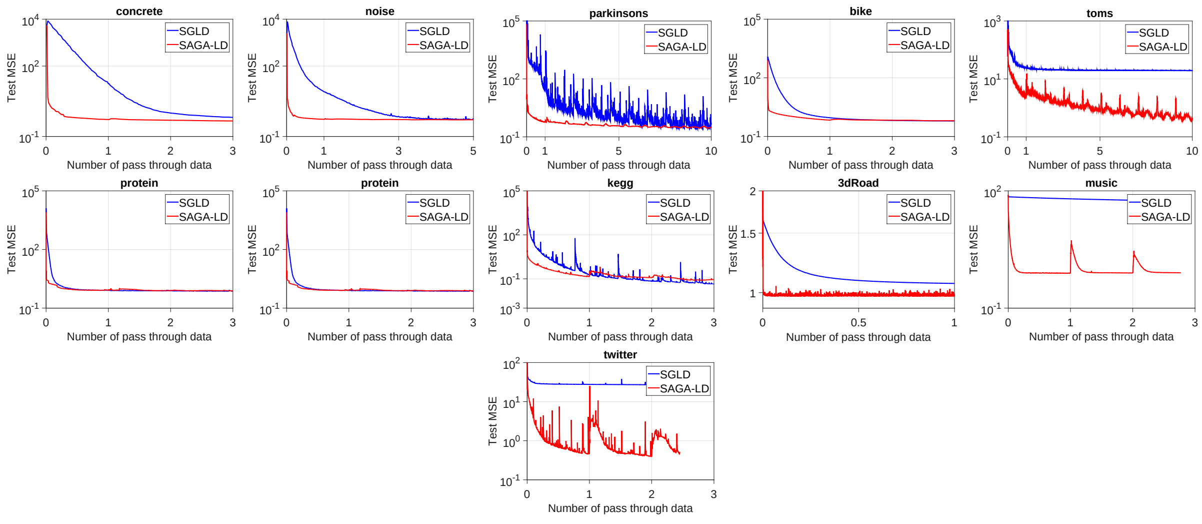


Figure 7.4: Performance comparison of SGLD and SAGA-LD on regression task. The x-axis and y-axis represent the number of pass through the entire data and average test MSE respectively.

7.11 Other Lemmatta

We state few useful and well-known lemmas in this section.

Lemma 7.11.1. *For random variables $z_1, \ldots, z_r$ are independent and mean 0, we have*

$$\mathbb{E}\left[\|z_1 + ... + z_r\|^2\right] = \mathbb{E}\left[\|z_1\|^2 + ... + \|z_r\|^2\right].$$

Proof. We have the following:

$$\mathbb{E}\left[\|z_1 + ... + z_r\|^2\right] = \sum_{i,i=1}^{r} \mathbb{E}\left[z_i z_j\right] = \mathbb{E}\left[\|z_1\|^2 + ... + \|z_r\|^2\right].$$

The second equality follows from the fact that z_i's are independent and mean 0. □

Lemma 7.11.2. *For random variables $z_1, \ldots, z_r$, we have*

$$\mathbb{E}\left[\|z_1 + ... + z_r\|^2\right] \leq r\mathbb{E}\left[\|z_1\|^2 + ... + \|z_r\|^2\right].$$

Part II

Large-Scale Empirical Risk Minimization

Chapter 8

Asynchronous Stochastic Variance Reduced Algorithms for ERM

8.1 Introduction

In this chapter, we turn our attention to asynchronous stochastic algorithms for convex finite-sum problems of the form:

$$\min_{x \in \mathbb{R}^d} f(x) := \tfrac{1}{n} \sum_{i=1}^{n} f_i(x). \tag{8.1}$$

There has been a steep rise in recent work on "variance reduced" stochastic gradient algorithms for convex finite-sum problems [33, 34, 53, 71, 74, 75, 153, 156, 173]. Under strong convexity assumptions such variance reduced (VR) stochastic algorithms attain better convergence rates (in expectation) than stochastic gradient descent (SGD) [110, 148], both in theory and practice.[1] The key property of these VR algorithms is that by exploiting problem structure and by making suitable space-time tradeoffs, they reduce the variance incurred due to stochastic gradients. This variance reduction has powerful consequences: it bestows VR stochastic methods with linear convergence rates, and thereby circumvents slowdowns that usually hit SGD.

Although these advances have great value in general, for large-scale problems we still require parallel or distributed processing. And in this setting, asynchronous variants of SGD remain indispensable [3, 35, 84, 128, 162, 182]. Therefore, a key question is how to extend the synchronous finite-sum VR algorithms to asynchronous parallel and distributed settings.

We answer one part of this question by developing new asynchronous parallel stochastic gradient methods that provably converge at a linear rate for smooth strongly convex finite-sum problems. Our methods are inspired by the influential SVRG [71], S2GD [74], SAG [153] and SAGA [33] family of algorithms. We list our contributions more precisely below.

[1]Though we should note that SGD also applies to the harder stochastic optimization problem $\min\ F(x) = \mathbb{E}[f(x;\xi)]$, which need not be a finite-sum.

Contributions. This chapter has two core components: (i) a formal general framework for variance reduced stochastic methods based on discussions in [33]; and (ii) asynchronous parallel VR algorithms within the framework. The general framework presents a formal unifying view of several VR methods (e.g., it includes SAGA and SVRG as special cases) while expressing key algorithmic and practical tradeoffs concisely. Thus, it yields a broader understanding of VR methods, which helps us obtain *asynchronous parallel* variants of VR methods. Under settings common to machine learning problems, our parallel algorithms attain speedups that scale near linearly with the number of processors. As a concrete illustration, we present a specialization to an asynchronous SVRG-like method. We compare this specialization with non-variance reduced asynchronous SGD methods, and observe strong empirical speedups that agree with the theory.

Related work. As already mentioned, our work is closest to (and generalizes) SAG [153], SAGA [33], SVRG [71] and S2GD [74], which are primal methods. Also closely related are dual methods such as SDCA [156] and Finito [34], and in its convex incarnation MISO [99]; a more precise relation between these dual methods and VR stochastic methods is described in Defazio's thesis [31]. By their algorithmic structure, these VR methods trace back to classical non-stochastic incremental gradient algorithms [18], but by now it is well-recognized that randomization helps obtain much sharper convergence results (in expectation). Proximal [173] and accelerated VR methods have also been proposed [116, 155]; we leave a study of such variants of our framework as future work. Finally, there is recent work on lower-bounds for finite-sum problems [2].

Within asynchronous SGD algorithms, both parallel [128] and distributed [3, 107] variants are known. In this chapter, we focus our attention on the parallel setting. A different line of methods is that of (primal) coordinate descent methods, and their parallel and distributed variants [90, 92, 113, 144]. Our asynchronous methods share some structural assumptions with these methods. Finally, the recent work [75] generalizes S2GD to the mini-batch setting, thereby also permitting parallel processing, albeit with more synchronization and allowing only small mini-batches.

8.2 A General Framework for VR Stochastic Methods

We focus on instances of (8.1) where the cost function $f(x)$ has an L-Lipschitz gradient, so that $\|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\|$, and it is λ-strongly convex, i.e., for all $x, y \in \mathbb{R}^d$,

$$f(x) \geq f(y) + \langle \nabla f(y), x - y \rangle + \tfrac{\lambda}{2}\|x - y\|^2. \tag{8.2}$$

In the first part of the chapter, we focus on strongly convex case and later extend our analysis to smooth convex functions in Section 8.4.

Inspired by the discussion on a general view of variance reduced techniques in [33], we now describe a formal general framework for variance reduction in stochastic gradient descent. We denote the collection $\{f_i\}_{i=1}^n$ of functions that make up f in (8.1) by $\mathcal{F}$. For our algorithm, we maintain an additional parameter $\alpha_i^t \in \mathbb{R}^d$ for each $f_i \in \mathcal{F}$.

Algorithm 20: GENERIC STOCHASTIC VARIANCE REDUCTION ALGORITHM

```
Data: $x^0 \in \mathbb{R}^d, \alpha_i^0 = x^0 \;\; \forall i \in [n] \triangleq \{1,\dots,n\}$, step size $\eta > 0$
Randomly pick a $I_T = \{i_0,\dots,i_T\}$ where $i_t \in \{1,\dots,n\} \; \forall\, t \in \{0,\dots,T\}$ ;
for $t = 0$ to $T$ do
    Update iterate as $x^{t+1} \leftarrow x^t - \eta \left( \nabla f_{i_t}(x^t) - \nabla f_{i_t}(\alpha_{i_t}^t) + \frac{1}{n}\sum_i \nabla f_i(\alpha_i^t) \right)$ ;
    $A^{t+1} = \textsc{ScheduleUpdate}(\{x^i\}_{i=0}^{t+1}, A^t, t, I_T)$ ;
end
return $x^T$
```

We use A^t to denote $\{\alpha_i^t\}_{i=1}^n$. The general iterative framework for updating the parameters is presented as Algorithm 20. Observe that the algorithm is still abstract, since it does not specify the subroutine SCHEDULEUPDATE. This subroutine determines the crucial update mechanism of $\{\alpha_i^t\}$ (and thereby of A^t). As we will see different schedules give rise to different fast first-order methods proposed in the literature. The part of the update based on A^t is the key for these approaches and is responsible for variance reduction.

Next, we provide different instantiations of the framework and construct a new algorithm derived from it. In particular, we consider incremental methods SAG [153], SVRG [71] and SAGA [33], and classic gradient descent GD for demonstrating our framework.

Figure 8.1 shows the schedules for the aforementioned algorithms. In case of SVRG, SCHEDULEUPDATE is triggered every m iterations (here m denotes precisely the number of inner iterations used in [71]); so A^t remains unchanged for the m iterations and all α_i^t are updated to the current iterate at the m^{th} iteration. For SAGA, unlike SVRG, A^t changes at the t^{th} iteration for all $t \in [T]$. This change is only to a single element of A^t, and is determined by the index i_t (the function chosen at iteration t). The update of SAG is similar to SAGA insofar that only one of the α_i is updated at each iteration. However, the update for A^{t+1} is based on i_{t+1} rather than i_t. This results in a biased estimate of the gradient, unlike SVRG and SAGA. Finally, the schedule for gradient descent is similar to SAG, except that all the α_i's are updated at each iteration. Due to the full update we end up with the exact gradient at each iteration. This discussion highlights how the scheduler determines the resulting gradient method.

To motivate the design of another schedule, let us consider the computational and storage costs of each of these algorithms. For SVRG, since we update A^t after every m iterations, it is enough to store a full gradient, and hence, the storage cost is $O(d)$. However, the running time is $O(nd)$ at each epoch and $O(d)$ at each iteration since we have to calculate the full gradient at end of each epoch. In contrast, both SAG and SAGA have high storage costs of $O(nd)$ and running time of $O(d)$ per iteration. Finally, GD has low storage cost since it needs to store the gradient at $O(d)$ cost, but very high computational costs of $O(nd)$ at *each* iteration.

It is interesting to note that when m is high (say greater than n), SVRG has low computational cost per iteration. However, as we will later see, this comes at the expense of slower convergence to the optimal solution. SAG and SAGA can converge faster by allowing us to update A^t more frequently, but at the cost of additional storage. The tradeoffs between the epoch size m, additional storage, frequency of updates, and the convergence to the optimal solution are still not completely resolved.

```
SVRG:ScheduleUpdate($\{x^i\}_{i=0}^{t+1}, A^t, t, I_T$)
for i = 1 to n do
    $\alpha_i^{t+1} = \mathbb{1}(m \mid t)x^t + \mathbb{1}(m \nmid t)\alpha_i^t$ ;
end
return $A^{t+1}$
```

```
SAGA:ScheduleUpdate($\{x^i\}_{i=0}^{t+1}, A^t, t, I_T$)
for i = 1 to n do
    $\alpha_i^{t+1} = \mathbb{1}(i_t = i)x^t + \mathbb{1}(i_t \neq i)\alpha_i^t$ ;
end
return $A^{t+1}$
```

```
SAG:ScheduleUpdate($\{x^i\}_{i=0}^{t+1}, A^t, t, I_T$)
for i = 1 to n do
    $\alpha_i^{t+1} = \mathbb{1}(i_{t+1} = i)x^{t+1} + \mathbb{1}(i_{t+1} \neq i)\alpha_i^t$ ;
end
return $A^{t+1}$
```

```
GD:ScheduleUpdate($\{x^i\}_{i=0}^{t+1}, A^t, t, I_T$)
for i = 1 to n do
    $\alpha_i^{t+1} = x^{t+1}$ ;
end
return $A^{t+1}$
```

Figure 8.1: SCHEDULEUPDATE function for SVRG (top left), SAGA (top right), SAG (bottom left) and GD (bottom right). While SVRG is epoch-based, rest of algorithms perform updates at each iteration. Here $a|b$ denotes that a divides b.

To design a new scheduler we combine the schedules of the above algorithms. We call this schedule *hybrid stochastic average gradient* (HSAG). Specifically, we use the schedules of SVRG and SAGA to develop HSAG. Consider $S \subseteq [n]$, the indices that follow SAGA schedule. We assume that rest of the indices follow an SVRG-like schedule with *schedule frequency* s_i for all $i \in \overline{S} \triangleq [n] \setminus S$. Figure 8.2 shows the corresponding update schedule of HSAG. If $S = [n]$ then HSAG is equivalent to SAGA, while at the other extreme, for $S = \emptyset$ and $s_i = m$ for all $i \in [n]$, it corresponds to SVRG. HSAG exhibits interesting storage, computational and convergence tradeoffs that depend on S. In general, while large cardinality of S likely incurs high storage costs, the computational cost per iteration is relatively low. On the other hand, when cardinality of S is small and s_i's are large, storage costs are low but the convergence typically slows down.

Before concluding our discussion on the general framework, we would like to draw the reader's attention to the advantages of studying Algorithm 20. First, note that Algorithm 20 provides a unifying framework for many incremental/stochastic gradient methods proposed in the literature. Second, and more importantly, it provides a generic platform for analyzing this class of algorithms. As we will see in Section 8.3, this helps us develop and analyze asynchronous versions for different finite-sum algorithms under a common umbrella. Finally, it provides a mechanism to derive new algorithms by designing more sophisticated schedules; as noted above, one such construction gives rise to HSAG.

```
HSAG:SCHEDULEUPDATE($x^t, A^t, t, I_T$)
for $i$ = 1 to $n$ do
    $\alpha_i^{t+1} = \begin{cases} \mathbb{1}(i_t = i)x^t + \mathbb{1}(i_t \neq i)\alpha_i^t & \text{if } i \in S \\ \mathbb{1}(s_i \mid t)x^t + \mathbb{1}(s_i \not\mid t)\alpha_i^t & \text{if } i \notin S \end{cases}$
end
return $A^{t+1}$
```

Figure 8.2: SCHEDULEUPDATE for HSAG.
This algorithm assumes access to the index set S and the schedule frequency vector s.

8.2.1 Convergence Analysis

In this section, we provide convergence analysis for Algorithm 20 with HSAG schedules. As observed earlier, SVRG and SAGA are special cases of this setup. Our analysis assumes unbiasedness of the gradient estimates at each iteration, so it does not encompass SAG. For ease of exposition, we assume that all $s_i = m$ for all $i \in [n]$. Since HSAG is epoch-based, our analysis focuses on the iterates obtained after each epoch. Similar to [71] (see Option II of SVRG in [71]), our analysis will be for the case where the iterate at the end of $(k+1)^{\text{st}}$ epoch, x^{km+m}, is replaced with an element chosen randomly from $\{x^{km}, \ldots, x^{km+m-1}\}$ with probability $\{p_1, \cdots, p_m\}$. For brevity, we use $\tilde{x}^k$ to denote the iterate chosen at the k^{th} *epoch*. We also need the following quantity for our analysis:

$$\tilde{G}_k \triangleq \frac{1}{n}\sum_{i\in S}\left(f_i(\alpha_i^{km}) - f_i(x^*) - \langle \nabla f_i(x^*), \alpha_i^{km} - x^*\rangle\right).$$

Theorem 8.2.1. *For any positive parameters $c, \beta, \kappa > 1$, step size η and epoch size m, we define the following quantities:*

$$\gamma = \kappa\left[1 - \left(1 - \frac{1}{\kappa}\right)^m\right]\left(2c\eta(1 - L\eta(1+\beta)) - \frac{1}{n} - \frac{2c}{\kappa\lambda}\right)$$
$$\theta = \max\left\{\left[\frac{2c}{\gamma\lambda}\left(1 - \frac{1}{\kappa}\right)^m + \frac{2Lc\eta^2}{\gamma}\left(1 + \frac{1}{\beta}\right)\kappa\left[1 - \left(1 - \frac{1}{\kappa}\right)^m\right]\right], \left(1 - \frac{1}{\kappa}\right)^m\right\}.$$

Suppose the probabilities $p_i \propto (1 - \frac{1}{\kappa})^{m-i}$, and that c, β, κ, step size η and epoch size m are chosen such that the following conditions are satisfied:

$$\frac{1}{\kappa} + 2Lc\eta^2\left(1 + \frac{1}{\beta}\right) \leq \frac{1}{n},\ \gamma > 0,\ \theta < 1.$$

Then, for iterates of Algorithm 20 under the HSAG *schedule, we have*

$$\mathbb{E}\left[f(\tilde{x}^{k+1}) - f(x^*) + \frac{1}{\gamma}\tilde{G}_{k+1}\right] \leq \theta\,\mathbb{E}\left[f(\tilde{x}^k) - f(x^*) + \frac{1}{\gamma}\tilde{G}_k\right].$$

As a corollary, we immediately obtain an expected linear rate of convergence for HSAG.

Corollary 8.2.1.1. *Note that $\tilde{G}_k \geq 0$ and therefore, under the conditions specified in Theorem 8.2.1 and $\bar{\theta} = \theta\left(1 + 1/\gamma\right) < 1$ we have*

$$\mathbb{E}\left[f(\tilde{x}^k) - f(x^*)\right] \leq \bar{\theta}^k \left[f(x^0) - f(x^*)\right].$$

We emphasize that there exist values of the parameters for which the conditions in Theorem 8.2.1 and Corollary 8.2.1.1 are easily satisfied. For instance, setting $\eta = 1/16(\lambda n + L)$, $\kappa = 4/\lambda\eta$, $\beta = (2\lambda n + L)/L$ and $c = 2/\eta n$, the conditions in Theorem 8.2.1 are satisfied for sufficiently large m. Additionally, in the high condition number regime of $L/\lambda = n$, we can obtain constant $\theta < 1$ (say 0.5) with $m = O(n)$ epoch size (similar to [33, 71]). This leads to ϵ accuracy in the objective function after $n \log(1/\epsilon)$ number of iterations.

8.3 Asynchronous Stochastic Variance Reduction

We are now ready to present asynchronous versions of the algorithms captured by our general framework. We first describe our setup before delving into the details of these algorithms. Our model of computation is similar to the ones used in Hogwild! [128] and AsySCD [92]. We assume a multicore architecture where each core makes stochastic gradient updates to a centrally stored vector x in an asynchronous manner. There are four key components in our asynchronous algorithm; these are briefly described below.

1. **Read**: Read iterate x and compute the gradient $\nabla f_{i_t}(x)$ for a randomly chosen i_t.
2. **Read schedule iterate**: Read the schedule iterate A and compute the gradients required for update in Algorithm 20.
3. **Update**: Update the iterate x with the update in Algorithm 20.
4. **Schedule Update**: Run a scheduler update for updating A.

Each processor repeatedly runs these procedures concurrently, without any synchronization. Hence, x may change in between Step 1 and Step 3. Similarly, A may change in between Steps 2 and 4. In fact, the states of iterates x and A can correspond to different time-stamps. We maintain a global counter t to track the number of updates successfully executed. We use $D(t) \in [t]$ and $D'(t) \in [t]$ to denote the particular x-iterate and A-iterate used for evaluating the update at the t^{th} iteration. We assume that the delay in between the time of evaluation and updating is bounded by a non-negative integer τ, i.e., $t - D(t) \leq \tau$ and $t - D'(t) \leq \tau$. The bound on the staleness captures the degree of parallelism in the method: such parameters are typical in asynchronous systems (see e.g., [92, 128]). Furthermore, we also assume that the system is synchronized after every epoch i.e., $D(t) \geq km$ for $t \geq km$. We would like to emphasize that the assumption is not strong since such a synchronization needs to be done only once per epoch.

For the purpose of our analysis, We assume a read consistent model. In particular, our analysis requires that the vector x used for evaluation of gradients be a valid iterate that existed at some point in time. However, like Hogwild! our implementation is lock-free. We will revisit this point in Section 8.5.

8.3.1 Convergence Analysis

The key ingredients to the success of asynchronous algorithms for multicore stochastic gradient descent are sparsity and "disjointness" of the data matrix [128] . These assumptions are typically satisfied in a variety of machine learning problems where one often has sparse, high-dimensional data. We also exploit these properties of the data for our convergence analysis. More formally, let $\|x\|_i$ denote the norm of x with respect to non-zero coordinates of function f_i; then, the convergence depends on Δ, the smallest constant such that $\mathbb{E}_i[\|x\|_i^2] \leq \Delta\|x\|^2$. Intuitively, Δ denotes the average frequency with which a feature appears in the data matrix. We are interested in situations where $\Delta \ll 1$. As a warm up, let us first discuss convergence analysis for asynchronous SVRG. The general case is similar, but much more involved. Hence, it is instructive to first go through the analysis of asynchronous SVRG.

Theorem 8.3.1. *Suppose step size η, epoch size m are chosen such that the following condition holds:*

$$0 < \theta_s := \frac{\left(\frac{1}{\lambda\eta m} + 4L\left(\frac{\eta + L\Delta\tau^2\eta^2}{1-2L^2\Delta\eta^2\tau^2}\right)\right)}{\left(1 - 4L\left(\frac{\eta + L\Delta\tau^2\eta^2}{1-2L^2\Delta\eta^2\tau^2}\right)\right)} < 1.$$

Then, for the iterates of an asynchronous variant of Algorithm 20 with SVRG *schedule and probabilities $p_i = 1/m$ for all $i \in [m]$, we have*

$$\mathbb{E}[f(\tilde{x}^{k+1}) - f(x^*)] \leq \theta_s\, \mathbb{E}[f(\tilde{x}^k) - f(x^*)].$$

The bound obtained in Theorem 8.3.1 is useful when Δ is small. To see this, as earlier, consider the indicative case where $L/\lambda = n$. The synchronous version of SVRG obtains a convergence rate of $\theta = 0.5$ for step size $\eta = 0.1/L$ and epoch size $m = O(n)$. For the asynchronous variant of SVRG, by setting $\eta = 0.1/2(\max\{1, \Delta^{1/2}\tau\}L)$, we obtain a similar rate with $m = O(n + \Delta^{1/2}\tau n)$. To obtain this, set $\eta = \rho/L$ where $\rho = 0.1/2(\max\{1, \Delta^{1/2}\tau\})$ and $\theta_s = 0.5$. Then, a simple calculation gives the following:

$$\frac{m}{n} = \frac{2}{\rho}\left(\frac{1 - 2\Delta\tau^2\rho^2}{1 - 12\rho - 14\Delta\tau^2\rho^2}\right) \leq c' \max\{1, \Delta^{1/2}\tau\},$$

where c' is some constant. This follows from the fact that $\rho = 0.1/2(\max\{1, \Delta^{1/2}\tau\})$. Suppose $\tau < 1/\Delta^{1/2}$. Then we can achieve nearly the same guarantees as the synchronous version, but τ times faster since we are running the algorithm asynchronously. For example, consider the sparse setting where $\Delta = o(1/n)$; then it is possible to get near linear speedup when $\tau = o(n^{1/2})$. On the other hand, when $\Delta^{1/2}\tau > 1$, we can obtain a theoretical speedup of $1/\Delta^{1/2}$.

We finally provide the convergence result for the asynchronous algorithm in the general case. The proof is complicated by the fact that set A, unlike in SVRG, changes during the epoch. The key idea is that only a single element of A changes at each iteration. Furthermore, it can only change to one of the iterates in the epoch. This control provides a handle on the error obtained due to the staleness. The proof is relegated to the appendix.

Theorem 8.3.2. *For any positive parameters $c, \beta, \kappa > 1$, step size η and epoch size m, we define the following quantities:*

$$\zeta = \left(c\eta^2 + \left(1 - \frac{1}{\kappa}\right)^{-\tau} cL\Delta\tau^2\eta^3 \right),$$

$$\gamma_a = \kappa \left[1 - \left(1 - \frac{1}{\kappa}\right)^m \right] \left[2c\eta - 8\zeta L(1+\beta) - \frac{2c}{\kappa\lambda} - \frac{96\zeta L\tau}{n}\left(1 - \frac{1}{\kappa}\right)^{-\tau} - \frac{1}{n} \right],$$

$$\theta_a = \max\left\{ \left[\frac{2c}{\gamma_a\lambda}\left(1 - \frac{1}{\kappa}\right)^m + \frac{8\zeta L\left(1 + \frac{1}{\beta}\right)}{\gamma_a}\kappa\left[1 - \left(1 - \frac{1}{\kappa}\right)^m\right] \right], \left(1 - \frac{1}{\kappa}\right)^m \right\}.$$

Suppose probabilities $p_i \propto (1 - \frac{1}{\kappa})^{m-i}$, parameters β, κ, step-size η, and epoch size m are chosen such that the following conditions are satisfied:

$$\frac{1}{\kappa} + 8\zeta L\left(1 + \frac{1}{\beta}\right) + \frac{96\zeta L\tau}{n}\left(1 - \frac{1}{\kappa}\right)^{-\tau} \le \frac{1}{n},\ \eta^2 \le \left(1 - \frac{1}{\kappa}\right)^{m-1}\frac{1}{12L^2\Delta\tau^2},\ \gamma_a > 0,\ \theta_a < 1.$$

Then, for the iterates of asynchronous variant of Algorithm 20 with HSAG *schedule we have*

$$\mathbb{E}\left[f(\tilde{x}^{k+1}) - f(x^*) + \frac{1}{\gamma_a}\tilde{G}_{k+1} \right] \le \theta_a \mathbb{E}\left[f(\tilde{x}^k) - f(x^*) + \frac{1}{\gamma_a}\tilde{G}_k \right].$$

Corollary 8.3.2.1. *Note that $\tilde{G}_k \ge 0$ and therefore, under the conditions specified in Theorem 8.3.2 and $\bar{\theta}_a = \theta_a\left(1 + 1/\gamma_a\right) < 1$, we have*

$$\mathbb{E}\left[f(\tilde{x}^k) - f(x^*) \right] \le \bar{\theta}_a^k \left[f(x^0) - f(x^*) \right].$$

By using step size normalized by $\Delta^{1/2}\tau$ (similar to Theorem 8.3.1) and parameters similar to the ones specified after Theorem 8.2.1 we can show speedups similar to the ones obtained in Theorem 8.3.1.

8.4 Non-strongly Convex Case

We extend our analysis to the case of non-strongly convex function in this section. The first part of our analysis deals with convergence of HSAG when the function is just convex. Recall that such an analysis also provides convergence analysis for methods such as SVRG and SAGA because they are merely extreme cases of HSAG.

The following is the main convergence result for the convex case. The key consequence of the result is that we obtain convergence rate of $O(1/T)$ as opposed to rate of

$O(1/\sqrt{T})$ obtained through SGD. All results will be based on the average of the iterates across T iterations (denoted by x_a^T). In all our results below we assume that T is a multiple of m. We expand function f as $f(x) = g(x) + h(x)$ where $g(x) = \frac{1}{n}\sum_{i \in S} f_i(x)$ and $h(x) = \frac{1}{n}\sum_{i \notin S} f_i(x)$.

Theorem 8.4.1. *For any positive parameters c, β, step size η and epoch size m, we define the following quantity:*

$$P_0' = \|x^0 - x^*\|^2 + D_g(x^0, x^*) + 2mLc\eta^2\left(1 + \frac{1}{\beta}\right)\mathbb{E}\left[D_h(\tilde{x}^k, x^*)\right].$$

Suppose probabilities $p_i = 1/n$, parameters β, κ, step-size η, and epoch size m are chosen such that the following conditions are satisfied:

$$2Lc\eta^2\left(1 + \frac{1}{\beta}\right) \leq \frac{1}{n},$$
$$2c\eta(1 - L\eta(1+\beta)) - \frac{1}{n} \geq 2Lc\eta^2\left(1 + \frac{1}{\beta}\right)$$

Then, for the iterates of Algorithm 20 with HSAG *schedule we have*

$$\mathbb{E}\left[f(x_a^T) - f(x^*)\right] \leq \frac{P_0}{T\left(2c\eta(1 - L\eta(1+\beta)) - \frac{1}{n} - 2Lc\eta^2\left(1 + \frac{1}{\beta}\right)\right)}.$$

Unlike the previous analysis of SVRG for non-strongly convex case that reduces to the strongly convex by adding a quadratic perturbation, our result provides a direct sublinear convergence rate. Finally, we also extend our analysis for non-strongly convex problems to the asynchronous setting. In particular, we show that with bounded delay, we can obtain performance gains similar to those obtained in the strongly convex case in Section 8.3.1.

Theorem 8.4.2. *For any positive parameters c, β, step size η and epoch size m, we define the following quantities:*

$$\zeta' = \left(c\eta^2 + cL\Delta\tau^2\eta^3\right),$$
$$P_0' = \|x^0 - x^*\|^2 + D_g(x^0, x^*) + 8m\zeta' L\left(1 + \frac{1}{\beta}\right)\mathbb{E}\left[D_h(x^0, x^*)\right].$$

Suppose probabilities $p_i = 1/n$, parameters β and κ, step-size η, and epoch size m are chosen such that the following conditions are satisfied:

$$8\zeta' L\left(1 + \frac{1}{\beta}\right) + \frac{96\zeta' L\tau}{n} \leq \frac{1}{n},\ \eta^2 \leq \frac{1}{12L^2\Delta\tau^2},$$
$$2c\eta - 8\zeta' L(1+\beta) - \frac{96\zeta' L\tau}{n} - \frac{1}{n} \geq 8\zeta' L\left(1 + \frac{1}{\beta}\right)$$

Then, for the iterates of asynchronous variant of Algorithm 20 with HSAG *schedule we have*

$$\mathbb{E}\left[f(x_a^T) - f(x^*)\right] \leq \frac{P_0'}{T\left[2c\eta - 8\zeta' L(1+\beta) - \frac{96\zeta' L\tau}{n} - \frac{1}{n} - 8\zeta' L\left(1+\frac{1}{\beta}\right)\right]}.$$

Before ending our discussion on the theoretical analysis, we would like to highlight an important point. Our emphasis throughout the chapter was on generality. While the results are presented here in full generality, one can obtain stronger results in specific cases. For example, in the case of SAGA, one can obtain *per iteration* convergence guarantees (see [33]) rather than those corresponding to *per epoch* presented in the chapter. Also, SAGA can be analyzed without any additional synchronization per epoch. However, there is no qualitative difference in these guarantees accumulated over the epoch. Furthermore, in this case, our analysis for both synchronous and asynchronous cases can be easily modified to obtain convergence properties similar to those in [33].

8.5 Experiments

We present our empirical results in this section. For our experiments, we study the problem of binary classification via l_2-regularized logistic regression. More formally, we are interested in the following optimization problem:

$$\min_x \frac{1}{n}\sum_{i=1}^{n}\left(\log(1+\exp(y_i z_i^\top x)) + \lambda\|x\|^2\right), \tag{8.3}$$

where $z_i \in \mathbb{R}^d$ and y_i is the corresponding label for each $i \in [n]$. In all our experiments, we set $\lambda = 1/n$. Note that such a choice leads to high condition number.

Since we are interested in sparse datasets, simply taking $f_i(x) = \log(1+\exp(y_i z_i^\top x)) + \lambda\|x\|^2$ is not efficient as it requires updating the whole vector x at each iteration. This is due to the regularization term in each of the f_i's. Instead, similar to [128], we rewrite problem in (8.3) as follows:

$$\min_x \frac{1}{n}\sum_{i=1}^{n}\left(\log(1+\exp(y_i z_i^\top x)) + \lambda \sum_{j\in nz(z_i)} \frac{\|x_j\|^2}{d_j}\right),$$

where $nz(z)$ represents the non-zero components of vector z, and $d_j = \sum_i \mathbb{1}(j \in nz(z_i))$. While this leads to sparse gradients at each iteration, updates in SVRG are still dense due to the part of the update that contains $\sum_i \nabla f_i(\alpha_i)/n$. This problem can be circumvented by using a 'just-in-time' update scheme similar to the one mentioned in [153]. First, recall that for SVRG, $\sum_i \nabla f_i(\alpha_i)/n$ does not change during an epoch (see Figure 8.1). Therefore, during the $(k+1)^{\text{st}}$ epoch we have the following relationship:

$$x^t = \left[\tilde{x}^k - \eta\sum_{j=km}^{t-1}(f_{i_j}(x^j) - f_{i_j}(\tilde{x}^k))\right] - \left[\frac{\eta(t-km)}{n}\sum_{i=1}^{n} f_i(\tilde{x}^k)\right].$$

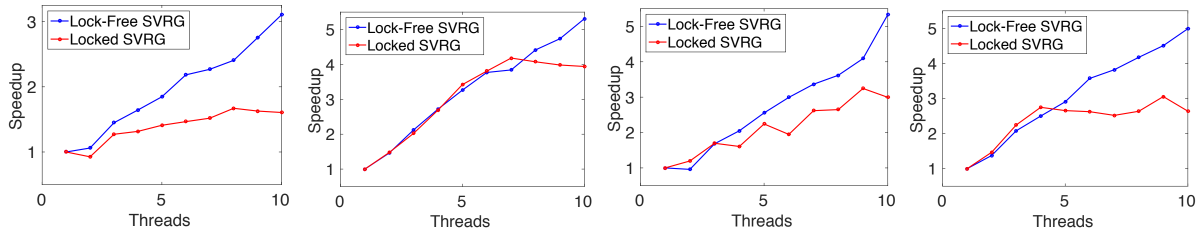


Figure 8.3: l_2-regularized logistic regression. Speedup curves for Lock-Free SVRG and Locked SVRG on rcv1 (left), real-sim (left center), news20 (right center) and url (right) datasets. We report the speedup achieved by increasing the number of threads.

We maintain each bracketed term separately. The updates to the first term in the above equation are sparse while those to the second term are just a simple scalar addition, since we already maintain the average gradient $\sum_{i=1}^{n} f_i(\tilde{x}^k)/n$. When the gradient of f_{i_t} at x^t is needed, we *only* calculate components of x^t required for f_{i_t} by aggregating these two terms. Hence, each step of this update procedure can be implemented in a way that respects sparsity of the data.

We evaluate the following algorithms for our experiments:

- **Lock-Free SVRG**: This is the lock-free asynchronous variant of Algorithm 20 using SVRG schedule; all threads can read and update the parameters with any synchronization. Parameter updates are performed through atomic *compare-and-swap* instruction facilitated by modern processors [128]. A constant step size that gives the best convergence is chosen for the dataset.
- **Locked SVRG**: This is the locked version of the asynchronous variant of Algorithm 20 using SVRG schedule. In particular, we use a *concurrent read exclusive write* locking model, where all threads can read the parameters but only one threads can update the parameters at a given time. The step size is chosen similar to Lock-Free SVRG.
- **Lock-Free SGD**: This is the lock-free asynchronous variant of the SGD algorithm (see [128]). We compare two different versions of this algorithm: (i) SGD with constant step size (referred to as CSGD). (ii) SGD with decaying step size $\eta_0\sqrt{\sigma_0/(t+\sigma_0)}$ (referred to as DSGD), where constants η_0 and σ_0 specify the scale and speed of decay. For each of these versions, step size is tuned for each dataset to give the best convergence progress.

All the algorithms were implemented in C++. The linear algebra operations are mainly performed using eigen3[2]. We run our experiments on datasets from LIBSVM website[3]. Similar to [173], we normalize each example in the dataset so that $\|z_i\|_2 = 1$ for all $i \in [n]$. Such a normalization leads to an upper bound of 0.25 on the Lipschitz constant of the gradient of f_i. The epoch size m is chosen as $2n$ (as recommended in [71]) in all our experiments. In the first experiment, we compare the speedup achieved by our asynchronous algorithm. To this end, for each dataset we first measure the time required

[2]http://eigen.tuxfamily.org/
[3]http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html

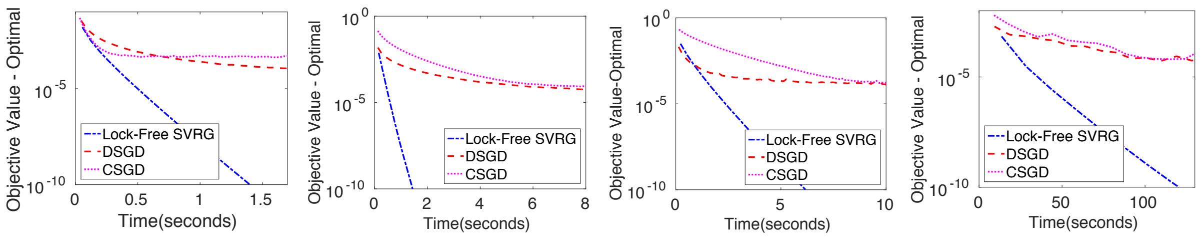


Figure 8.4: l_2-regularized logistic regression. Training loss residual $f(x) - f(x^*)$ versus time plot of Lock-Free SVRG, DSGD and CSGD on rcv1 (left), real-sim (left center), news20 (right center) and url (right) datasets. The experiments are parallelized over 10 cores.

for the algorithm to each an accuracy of 10^{-10} (i.e., $f(x) - f(x^*) < 10^{-10}$). The speedup with P threads is defined as the ratio of the runtime with a single thread to the runtime with P threads. Results in Figure 8.3 show the speedup on various datasets. As seen in the figure, we achieve significant speedups for all the datasets. Not surprisingly, the speedup achieved by Lock-free SVRG is much higher than ones obtained by locking. Furthermore, the lowest speedup is achieved for rcv1 dataset. Similar speedup behavior was reported for this dataset in [128]. It should be noted that this dataset is not sparse and hence, is a bad case for the algorithm (similar to [128]).

For the second set of experiments we compare the performance of Lock-Free SVRG with stochastic gradient descent. In particular, we compare with the variants of stochastic gradient descent, DSGD and CSGD, described earlier in this section. It is well established that the performance of variance reduced stochastic methods is better than that of SGD. We would like to empirically verify that such benefits carry over to the asynchronous variants of these algorithms. Figure 8.4 shows the performance of Lock-Free SVRG, DSGD and CSGD. Since the computation complexity of each epoch of these algorithms is different, we directly plot the objective value versus the runtime for each of these algorithms. We use 10 cores for comparing the algorithms in this experiment. As seen in the figure, Lock-Free SVRG outperforms both DSGD and CSGD. The performance gains are qualitatively similar to those reported in [71] for the synchronous versions of these algorithms. It can also be seen that the DSGD, not surprisingly, outperforms CSGD in all the cases. In our experiments, we observed that Lock-Free SVRG, in comparison to SGD, is relatively much less sensitive to the step size and more robust to increasing threads.

8.6 Discussion & Future Work

In this chapter, we presented a unifying framework based on [33], that captures many popular variance reduction techniques for stochastic gradient descent. We use this framework to develop a simple hybrid variance reduction method. The primary purpose of the framework, however, was to provide a common platform to analyze various variance reduction techniques. To this end, we provided convergence analysis for the framework under certain conditions. More importantly, we propose an asynchronous algorithm for the framework with provable convergence guarantees. The key consequence of our approach is that we obtain asynchronous variants of several algorithms like SVRG, SAGA and S2GD. Our asynchronous algorithms exploits sparsity in the data to obtain near linear speedup in settings that are typically encountered in machine learning.

For future work, it would be interesting to perform an empirical comparison of various schedules. In particular, it would be worth exploring the space-time-accuracy tradeoffs of these schedules. We would also like to analyze the effect of these tradeoffs on the asynchronous variants.

Appendix: Omitted Proofs

Notation: We use D_f to denote the Bregman divergence (defined below) for function f.

$$D_f(x, y) = f(x) - f(y) - \langle \nabla f(y), x - y \rangle.$$

For ease of exposition, we use $\mathbb{E}[X]$ to denote the expectation the random variable X with respect to indices $\{i_1, \ldots, i_t\}$ when X depends on just these indices up to step t. This dependence will be clear from the context. We use $\mathbb{1}$ to denote the indicator function.

We would like to clarify the definition of x^{km} here. As noted in the text, for analysis of strongly convex functions, we assume that x^{km+m} is replaced with the average of the iterates in the k^{th} epoch at the end of the epoch. However, whenever x^{km} appears in the analysis, it represents the iterate before it is replaced by the average of the iterates.

Proof of Theorem 8.2.1

Proof. We expand function f as $f(x) = g(x) + h(x)$ where $g(x) = \frac{1}{n}\sum_{i \in S} f_i(x)$ and $h(x) = \frac{1}{n}\sum_{i \notin S} f_i(x)$. Let the present epoch be $k+1$. We define the following:

$$\begin{aligned}
v^t &= \frac{1}{\eta}(x^{t+1} - x^t) = -\left[\nabla f_{i_t}(x^t) - \nabla f_{i_t}(\alpha^t_{i_t}) + \frac{1}{n}\sum_i \nabla f_i(\alpha^t_i)\right] \\
G_t &= \frac{1}{n}\sum_{i \in S} \left(f_i(\alpha^t_i) - f_i(x^*) - \langle \nabla f_i(x^*), \alpha^t_i - x^* \rangle\right) \\
R_t &= \mathbb{E}\left[c\|x^t - x^*\|^2 + G_t\right].
\end{aligned}$$

We first observe that $\mathbb{E}[v^t] = -\nabla f(x^t)$. This follows from the unbiasedness of the gradient at each iteration. Using this observation, we have the following:

$$\begin{aligned}
\mathbb{E}[R_{t+1}] &= \mathbb{E}[c\|x^{t+1} - x^*\|^2 + G_{t+1}] = \mathbb{E}[c\|x^t + \eta v^t - x^*\|^2 + G_{t+1}] \\
&= c\mathbb{E}\left[\|x^t - x^*\|^2\right] + c\eta^2\mathbb{E}\left[\|v^t\|^2\right] + 2c\eta\mathbb{E}\left[\langle x^t - x^*, v^t\rangle\right] + \mathbb{E}[G_{t+1}] \\
&\leq c\mathbb{E}\left[\|x^t - x^*\|^2\right] + c\eta^2\mathbb{E}\left[\|v^t\|^2\right] - 2c\eta\mathbb{E}\left[f(x^t) - f(x^*)\right] + \mathbb{E}[G_{t+1}]. \quad (8.4)
\end{aligned}$$

The last step follows from convexity of f and the unbiasedness of v^t. We have the following relationship between G_{t+1} and G_t.

$$\begin{aligned}
\mathbb{E}[G_{t+1}] &= \left(1 - \frac{1}{n}\right)\mathbb{E}\left[G_t\right] + \frac{1}{n}\mathbb{E}\left[\frac{1}{n}\sum_{i\in S}\left(f_i(x^t) - f_i(x^*) - \langle\nabla f_i(x^*), x^t - x^*\rangle\right)\right] \\
&= \left(1 - \frac{1}{n}\right)\mathbb{E}\left[G_t\right] + \frac{1}{n}\mathbb{E}[D_g(x^t, x^*)]. \quad (8.5)
\end{aligned}$$

This follows from the definition of the schedule of HSAG for indices in S. Substituting the above relationship in Equation (8.4) we get the following.

$$\begin{aligned}
R_{t+1} &\leq R_t + c\eta^2\mathbb{E}\left[\|v^t\|^2\right] - 2c\eta\mathbb{E}\left[f(x^t) - f(x^*)\right] - \frac{1}{n}\mathbb{E}[G_t] + \frac{1}{n}\mathbb{E}[D_g(x^t, x^*)] \\
&\leq \left(1 - \frac{1}{\kappa}\right)R_t + \frac{c}{\kappa}\mathbb{E}[\|x^t - x^*\|^2] + c\eta^2\mathbb{E}\left[\|v^t\|^2\right] - 2c\eta\mathbb{E}\left[f(x^t) - f(x^*)\right] \\
&\qquad + \left(\frac{1}{\kappa} - \frac{1}{n}\right)\mathbb{E}[G_t] + \frac{1}{n}\mathbb{E}[D_g(x^t, x^*)] \\
&:= \left(1 - \frac{1}{\kappa}\right)R_t + b_t.
\end{aligned}$$

We describe the bounds for b_t (defined below).

$$\begin{aligned}
b_t = \frac{c}{\kappa}\underbrace{\mathbb{E}[\|x^t - x^*\|^2]}_{T_1} + c\eta^2\underbrace{\mathbb{E}\left[\|v^t\|^2\right]}_{T_2} - 2c\eta\mathbb{E}\left[f(x^t) - f(x^*)\right] \\
+ \left(\frac{1}{\kappa} - \frac{1}{n}\right)\mathbb{E}[G_t] + \frac{1}{n}\mathbb{E}[D_g(x^t, x^*)].
\end{aligned}$$

The terms T_1 and $T2$ can be bounded in the following fashion:

$$
\begin{aligned}
T_1 &= \mathbb{E}[\|x^t - x^*\|^2] \leq \frac{2}{\lambda}\mathbb{E}[f(x^t) - f(x^*)] \\
T_2 &= \mathbb{E}\left[\|v^t\|^2\right] \leq \left(1 + \frac{1}{\beta}\right)\mathbb{E}\left[\|\nabla f_{i_t}(\alpha_{i_t}^t) - \nabla f_{i_t}(x^*)\|^2\right] + (1+\beta)\mathbb{E}\left[\|\nabla f_{i_t}(x^t) - \nabla f_{i_t}(x^*)\|^2\right] \\
&\leq \frac{2L}{n}\left(1 + \frac{1}{\beta}\right)\mathbb{E}\sum_i \left[f_i(\alpha_i^t) - f(x^*) - \langle \nabla f_i(x^*), \alpha_i^t - x^*\rangle\right] \\
&\quad + \frac{2L}{n}(1+\beta)\mathbb{E}\sum_i \left[f_i(x^t) - f(x^*)\right] \\
&\leq 2L\left(1 + \frac{1}{\beta}\right)\mathbb{E}\left[G_t + D_h(\tilde{x}^k, x^*)\right] + 2L(1+\beta)\mathbb{E}[f(x^t) - f(x^*)].
\end{aligned}
$$

The bound on T_1 is due to strong convexity of f. The first inequality and second inequalities on T_2 directly follows from Lemma 3 of [33] and simple application of Lemma 8.8.1 respectively. The third inequality follows from the definition of G_t and the fact that $\alpha_i^t = \tilde{x}^k$ for all $i \notin S$ and $t \in \{km, \dots, km + m - 1\}$.

Substituting these bounds T_1 and T_2 in b_t, we get

$$
\begin{aligned}
b_t &\leq -\left[2c\eta - 2cL\eta^2(1+\beta) - \frac{2c}{\kappa\lambda}\right]\mathbb{E}\left[f(x^t) - f(x^*)\right] \\
&\quad + \left(\frac{1}{\kappa} + 2cL\eta^2\left(1 + \frac{1}{\beta}\right) - \frac{1}{n}\right)\mathbb{E}[G_t] + \frac{1}{n}\mathbb{E}[D_g(x^t, x^*)] \\
&\quad + 2cL\eta^2\left(1 + \frac{1}{\beta}\right)\mathbb{E}\left[D_h(\tilde{x}^k, x^*)\right] \\
&\leq -\left[2c\eta - 2cL\eta^2(1+\beta) - \frac{1}{n} - \frac{2c}{\kappa\lambda}\right]\mathbb{E}\left[f(x^t) - f(x^*)\right] \\
&\quad + \left(\frac{1}{\kappa} + 2cL\eta^2\left(1 + \frac{1}{\beta}\right) - \frac{1}{n}\right)\mathbb{E}[G_t] + 2cL\eta^2\left(1 + \frac{1}{\beta}\right)\mathbb{E}\left[D_h(\tilde{x}^k, x^*)\right] \\
&\leq -\left[2c\eta - 2cL\eta^2(1+\beta) - \frac{1}{n} - \frac{2c}{\kappa\lambda}\right]\mathbb{E}\left[f(x^t) - f(x^*)\right] + 2cL\eta^2\left(1 + \frac{1}{\beta}\right)\mathbb{E}\left[D_h(\tilde{x}^k, x^*)\right].
\end{aligned}
\tag{8.6}
$$

The second inequality follows from Lemma 8.8.2. In particular, we use the fact that $f(x) - f(x^*) = D_f(x, x^*)$ and $D_f(x, x^*) = D_g(x, x^*) + D_h(x, x^*) \geq D_g(x, x^*)$. The third inequality follows from the following for the choice of our parameters:

$$\frac{1}{\kappa} + 2Lc\eta^2\left(1 + \frac{1}{\beta}\right) \leq \frac{1}{n}.$$

Applying the recursive relationship on R_{t+1} for m iterations, we get

$$R_{km+m} \leq \left(1 - \frac{1}{\kappa}\right)^m \tilde{R}_k + \sum_{j=0}^{m-1}\left(1 - \frac{1}{\kappa}\right)^{m-1-j} b_{km+j}$$

where

$$\tilde{R}_k = \mathbb{E}\left[c\|\tilde{x}^k - x^*\|^2 + \tilde{G}_k\right].$$

Substituting the bound on b_t from Equation (8.6) in the above equation we get the following inequality:

$$\begin{aligned}
R_{km+m} \leq{}& \left(1 - \frac{1}{\kappa}\right)^m \tilde{R}_k \\
&- \sum_{j=0}^{m-1} \left(2c\eta(1 - L\eta(1+\beta)) - \frac{1}{n} - \frac{2c}{\kappa\lambda}\right)\left(1 - \frac{1}{\kappa}\right)^{m-1-j} \mathbb{E}\left[f(x^{km+j}) - f(x^*)\right] \\
&+ \sum_{j=0}^{m-1} \left(1 - \frac{1}{\kappa}\right)^{m-1-j} 2Lc\eta^2 \left(1 + \frac{1}{\beta}\right) \mathbb{E}\left[h(\tilde{x}^k) - h(x^*) - \langle \nabla h(x^*), \tilde{x}^k - x^* \rangle\right].
\end{aligned}$$

We now use the fact that $\tilde{x}^{k+1}$ is chosen randomly from $\{x^{km}, \ldots, x^{km+m-1}\}$ with probabilities proportional to $\{(1-1/\kappa)^{m-1}, \ldots, 1\}$ we have the following consequence of the above inequality:

$$\begin{aligned}
R_{km+m} &+ \kappa\left[1 - \left(1 - \frac{1}{\kappa}\right)^m\right]\left(2c\eta(1 - L\eta(1+\beta)) - \frac{1}{n} - \frac{2c}{\kappa\lambda}\right) \mathbb{E}\left[f(\tilde{x}^{k+1}) - f(x^*)\right] \\
&\leq \frac{2c}{\lambda}\left(1 - \frac{1}{\kappa}\right)^m \mathbb{E}\left[f(\tilde{x}^k) - f(x^*)\right] + \left(1 - \frac{1}{\kappa}\right)^m \mathbb{E}\left[\tilde{G}_k\right] \\
&\quad + 2Lc\eta^2\kappa\left[1 - \left(1 - \frac{1}{\kappa}\right)^m\right]\left(1 + \frac{1}{\beta}\right)\mathbb{E}\left[D_h(\tilde{x}^k, x^*)\right].
\end{aligned}$$

For obtaining the above inequality, we used strong convexity of f. Again, using the Bregman divergence based inequality (see Lemma 8.8.2)

$$f(x) - f(x^*) = D_f(x, x^*) = D_g(x, x^*) + D_h(x, x^*) \geq D_h(x, x^*),$$

we have the following inequality

$$\begin{aligned}
&R_{km+m} + \kappa\left[1 - \left(1 - \frac{1}{\kappa}\right)^m\right]\left(2c\eta(1 - L\eta(1+\beta)) - \frac{1}{n} - \frac{2c}{\kappa\lambda}\right)\mathbb{E}\left[f(\tilde{x}^{k+1}) - f(x^*)\right] \\
&\leq \left[\frac{2c}{\lambda}\left(1 - \frac{1}{\kappa}\right)^m + 2Lc\eta^2\kappa\left(1 + \frac{1}{\beta}\right)\left[1 - \left(1 - \frac{1}{\kappa}\right)^m\right]\right]\mathbb{E}\left[f(\tilde{x}^k) - f(x^*)\right] + \left(1 - \frac{1}{\kappa}\right)^m \mathbb{E}\left[\tilde{G}_k\right].
\end{aligned} \tag{8.7}$$

Introduce now the notation

$$\begin{aligned}
\gamma &= \kappa\left[1 - \left(1 - \frac{1}{\kappa}\right)^m\right]\left(2c\eta(1 - L\eta(1+\beta)) - \frac{1}{n} - \frac{2c}{\kappa\lambda}\right) \\
\theta &= \max\left\{\left[\frac{2c}{\gamma\lambda}\left(1 - \frac{1}{\kappa}\right)^m + \frac{2Lc\eta^2}{\gamma}\left(1 + \frac{1}{\beta}\right)\kappa\left[1 - \left(1 - \frac{1}{\kappa}\right)^m\right]\right], \left(1 - \frac{1}{\kappa}\right)^m\right\}.
\end{aligned}$$

Using this notation along with (8.7) we obtain the inequality

$$\mathbb{E}\left[f(\tilde{x}^{k+1}) - f(x^*) + \frac{1}{\gamma}\tilde{G}_{k+1}\right] \leq \theta\,\mathbb{E}\left[f(\tilde{x}^{k}) - f(x^*) + \frac{1}{\gamma}\tilde{G}_{k}\right],$$

where $\theta < 1$ is a constant that depends on the parameters used in the algorithm. □

Proof of Theorem 8.3.1

Proof. Let the present epoch be $k+1$. Recall that $D(t)$ denotes the iterate used in the t^{th} iteration of the algorithm. We define the following two sequences:

$$\begin{aligned} u^t &= -\left[\nabla f_{i_t}(x^{D(t)}) - \nabla f_{i_t}(\tilde{x}^k) + \nabla f(\tilde{x}^k)\right] \\ v^t &= -\left[\nabla f_{i_t}(x^{t}) - \nabla f_{i_t}(\tilde{x}^k) + \nabla f(\tilde{x}^k)\right]. \end{aligned}$$

Consider now

$$\mathbb{E}\|x^{t+1} - x^*\|^2 = \mathbb{E}\|x^t + \eta u^t - x^*\|^2 = \mathbb{E}\left[\|x^t - x^*\|^2 + \eta^2\|u^t\|^2 + 2\eta\langle x^t - x^*, u^t\rangle\right]. \tag{8.8}$$

We first bound the last term of the above inequality. We expand the term in the following manner:

$$\begin{aligned} \mathbb{E}\langle x^t - x^*, u^t\rangle &= \mathbb{E}\left[\langle x^* - x^t, \nabla f_{i_t}(x^{D(t)})\rangle\right] \\ &= \underbrace{\mathbb{E}\left[\langle x^* - x^{D(t)}, \nabla f_{i_t}(x^{D(t)})\rangle\right]}_{T_3} \\ &\quad + \underbrace{\sum_{d=D(t)}^{t-1} \mathbb{E}\left[\langle x^d - x^{d+1}, \nabla f_{i_t}(x^d)\rangle\right]}_{T_4} + \underbrace{\sum_{d=D(t)}^{t-1} \mathbb{E}\left[\langle x^d - x^{d+1}, \nabla f_{i_t}(x^{D(t)}) - \nabla f_{i_t}(x^d)\rangle\right]}_{T_5}. \end{aligned} \tag{8.9}$$

The first equality directly follows from the definition of u^t and its property of unbiasedness. The second step follows from simple algebraic calculations. Terms T_3 and T_4 can be bounded in the following way:

$$T_3 \leq \mathbb{E}[f_{i_t}(x^*) - f_{i_t}(x^{D(t)})]. \tag{8.10}$$

This bound directly follows from convexity of function f_{i_t}.

$$\begin{aligned} T_4 &= \sum_{d=D(t)}^{t-1} \mathbb{E}\left[\langle x^d - x^{d+1}, \nabla f_{i_t}(x^d)\rangle\right] \\ &\leq \sum_{d=D(t)}^{t-1} \mathbb{E}\left[f_{i_t}(x^d) - f_{i_t}(x^{d+1}) + \frac{L}{2}\|x^{d+1} - x^d\|_{i_t}^2\right] \\ &\leq \mathbb{E}\left[f_{i_t}(x^{D(t)}) - f_{i_t}(x^t)\right] + \frac{L\Delta}{2} \sum_{d=D(t)}^{t-1} \mathbb{E}\left[\|x^{d+1} - x^d\|^2\right]. \qquad (8.11) \end{aligned}$$

The first inequality follows Lipschitz continuity of ∇f_{i_t}. The second inequality follows from the definition of Δ. The last term T_5 can be bounded as follows.

$$\begin{aligned} T_5 &= \mathbb{E}\left[\sum_{d=D(t)}^{t-1} \langle x^d - x^{d+1}, \nabla f_{i_t}(x^{D(t)}) - \nabla f_{i_t}(x^d)\rangle\right] \\ &\leq \mathbb{E}\left[\sum_{d=D(t)}^{t-1} \|x^{d+1} - x^d\|_{i_t} \|\nabla f_{i_t}(x^{D(t)}) - \nabla f_{i_t}(x^d)\|\right] \\ &\leq \mathbb{E}\left[\sum_{d=D(t)}^{t-1} \|x^{d+1} - x^d\|_{i_t} \sum_{j=D(t)}^{d-1} \|\nabla f_{i_t}(x^{j+1}) - \nabla f_{i_t}(x^j)\|\right] \qquad (8.12) \end{aligned}$$

The first inequality follows from Cauchy-Schwartz inequality. The second inequality follows from repeated application of triangle inequality. Furthermore, the aforementioned bound on T_5 can be bounded in the following manner:

$$\begin{aligned} T_5 &\leq \mathbb{E}\left[\sum_{d=D(t)}^{t-1} \sum_{j=D(t)}^{d-1} \frac{L}{2}\left(\|x^{d+1} - x^d\|_{i_t}^2 + \|x^{j+1} - x^j\|_{i_t}^2\right)\right] \\ &\leq \frac{L\Delta(\tau - 1)}{2} \mathbb{E} \sum_{d=D(t)}^{t-1} \|x^{d+1} - x^d\|^2. \qquad (8.13) \end{aligned}$$

The first step uses Lipschitz continuity of the gradient ∇f_{i_t} combined with the AM-GM inequality. Finally, the last step can be obtained from the fact that the staleness in gradient is at most τ and the definition of Δ.

By combining the bounds on T_3, T_4 and T_5 in Equations (8.10), (8.11) and (8.13) respectively and substituting the sum in Equation (8.9), we get

$$\mathbb{E}\langle x^t - x^*, u^t\rangle \leq \mathbb{E}\left[f(x^*) - f(x^t) + \frac{L\Delta\tau}{2} \sum_{d=D(t)}^{t-1} \|x^{d+1} - x^d\|^2\right]. \qquad (8.14)$$

By substituting the above inequality in (8.8) and noting that $x^{d+1} - x^d = \eta u^d$, we get

$$\mathbb{E}\left[\|x^{t+1} - x^*\|^2\right] \leq \mathbb{E}\left[\|x^t - x^*\|^2 + \eta^2\|u^t\|^2 - 2\eta(f(x^t) - f(x^*)) + L\Delta\tau\eta^3 \sum_{d=D(t)}^{t-1} \|u^d\|^2\right]. \tag{8.15}$$

We next bound the term $\mathbb{E}[\|u^t\|^2]$ in terms of $\mathbb{E}\left[\|v^t\|^2\right]$ in the following way:

$$\begin{aligned}
\mathbb{E}\left[\|u^t\|^2\right] &\leq 2\mathbb{E}\left[\|u^t - v^t\|^2 + \|v^t\|^2\right] \\
&\leq 2\mathbb{E}\left[\|\nabla f_{i_t}(x^t) - \nabla f_{i_t}(x^{D(t)})\|^2\right] + 2\mathbb{E}\left\|v^t\|^2\right] \\
&\leq 2L^2\mathbb{E}\left[\|x^{d+1} - x^d\|_{i_t}^2\right] + 2\mathbb{E}\left[\|v^t\|^2\right] \\
&\leq 2L^2\tau \sum_{d=D(t)}^{t-1} \mathbb{E}\left[\|x^t - x^{D(t)}\|_{i_t}^2\right] + 2\mathbb{E}\left[\|v^t\|^2\right] \\
&\leq 2L^2\Delta\eta^2\tau \sum_{d=D(t)}^{t-1} \mathbb{E}\left[\|u^d\|^2\right] + 2\mathbb{E}\left[\|v^t\|^2\right].
\end{aligned}$$

The first step follows from AM-GM inequality. The second inequality follows from Lipschitz continuity of the gradient. The third step is a simple application of Cauchy-Schwarz. Adding the above inequalities from $t = km$ to $t = km + m - 1$, we get

$$\begin{aligned}
\sum_{t=km}^{km+m-1} \mathbb{E}\left[\|u^t\|^2\right] &\leq \sum_{t=km}^{km+m-1} \left[2L^2\Delta\eta^2\tau \sum_{d=D(t)}^{t-1} \mathbb{E}\left[\|u^d\|^2\right] + 2\mathbb{E}\left[\|v^t\|^2\right]\right] \\
&\leq 2L^2\Delta\eta^2\tau^2 \sum_{t=km}^{km+m-1} \mathbb{E}\left[\|u^t\|^2\right] + 2\sum_{t=km}^{km+m-1} \mathbb{E}\left[\|v^t\|^2\right].
\end{aligned}$$

Here we again used the fact that the delay in the gradients is at most τ. From the above inequality, we get

$$\sum_{t=km}^{km+m-1} \mathbb{E}\left[\|u^t\|^2\right] \leq \frac{2}{(1 - 2L^2\Delta\eta^2\tau^2)} \sum_{t=km}^{km+m-1} \mathbb{E}\left[\|v^t\|^2\right]. \tag{8.16}$$

Adding Equation (8.15) from $t = km$ to $t = km + m - 1$ and substituting Equation (8.16) in the resultant, we get

$$\begin{aligned}
\mathbb{E}\left[\|x^{km+m} - x^*\|^2\right] &\leq \mathbb{E}\left[\|\tilde{x}^k - x^*\|^2 + (\eta^2 + L\Delta\tau^2\eta^3) \sum_{t=km}^{km+m-1} \|u^t\|^2 - \sum_{t=km}^{km+m-1} 2\eta(f(x^t) - f(x^*))\right] \\
&\leq \mathbb{E}\left[\|\tilde{x}^k - x^*\|^2 + 2\left(\frac{\eta^2 + L\Delta\tau^2\eta^3}{1 - 2L^2\Delta\eta^2\tau^2}\right) \sum_{t=km}^{km+m-1} \|v^t\|^2 - \sum_{t=km}^{km+m-1} 2\eta(f(x^t) - f(x^*))\right].
\end{aligned}$$

The first step follows from telescopy sum and the definition of $\tilde{x}^k$. From Lemma 3 of [33] (also see [71]), we have

$$\mathbb{E}[\|v^t\|^2] \leq 4L\mathbb{E}\left[f(x^t) - f(x^*) + f(\tilde{x}^k) - f(x^*)\right].$$

Substituting this in the inequality above , we get the following bound:

$$\begin{aligned}
&\left(2\eta - 8L\left(\frac{\eta^2 + L\Delta\tau^2\eta^3}{1 - 2L^2\Delta\eta^2\tau^2}\right)\right) m\mathbb{E}[f(\tilde{x}^{k+1}) - f(x^*)] \\
&\leq \left(\frac{2}{\mu} + 8L\left(\frac{\eta^2 + L\Delta\tau^2\eta^3}{1 - 2L^2\Delta\eta^2\tau^2}\right) m\right)\mathbb{E}[f(\tilde{x}^k) - f(x^*)].
\end{aligned}$$

□

Proof of Theorem 8.3.2

Proof. Let the present epoch be $k+1$. For simplicity, we assume that the iterates x and A used in the each iteration are from the same time step (index) i.e., $D(t) = D'(t)$ for all $t \in T$. Recall that $D(t)$ and $D'(t)$ denote the index used in the t^{th} iteration of the algorithm. Our analysis can be extended to the case of $D(t) \neq D'(t)$ in a straightforward albeit tedious manner. We expand function f as $f(x) = g(x) + h(x)$ where $g(x) = \frac{1}{n}\sum_{i \in S} f_i(x)$ and $h(x) = \frac{1}{n}\sum_{i \notin S} f_i(x)$. We define

$$\begin{aligned}
u^t &= \frac{1}{\eta}(x^{t+1} - x^t) = -\left[\nabla f_{i_t}(x^{D(t)}) - \nabla f_{i_t}(\alpha_{i_t}^{D(t)}) + \frac{1}{n}\sum_i \nabla f_i(\alpha_i^{D(t)})\right] \\
v^t &= -\left[\nabla f_{i_t}(x^t) - \nabla f_{i_t}(\alpha_{i_t}^t) + \frac{1}{n}\sum_i \nabla f_i(\alpha_i^t)\right].
\end{aligned}$$

We use the same Lyapunov function used in Theorem 8.2.1. We recall the following definitions:

$$\begin{aligned}
G_t &= \frac{1}{n}\sum_{i \in S}\left(f_i(\alpha_i^t) - f_i(x^*) - \langle \nabla f_i(x^*), \alpha_i^t - x^*\rangle\right) \\
R_t &= \mathbb{E}\left[c\|x^t - x^*\|^2 + G_t\right].
\end{aligned}$$

Using unbiasedness of the gradient we have $\mathbb{E}[u^t] = -\nabla f(x^{D(t)})$ and $\mathbb{E}[v^t] = -\nabla f(x^t)$. Using this observation, we have the following:

$$\begin{aligned}
c\mathbb{E}[\|x^{t+1} - x^*\|^2] &= c\mathbb{E}[\|x^t + \eta u^t - x^*\|^2] \\
&= c\mathbb{E}\left[\|x^t - x^*\|^2\right] + c\eta^2 \underbrace{\mathbb{E}\left[\|u^t\|^2\right]}_{T_6} + 2c\eta \underbrace{\mathbb{E}\left[\langle x^t - x^*, u^t\rangle\right]}_{T_7}. \qquad (8.17)
\end{aligned}$$

The last step follows from convexity of f and the unbiasedness of u^t. We further bound term T_6 in the following manner:

$$T_6 = \mathbb{E}\left[\|u^t\|^2\right] \leq 2\mathbb{E}\left[\|u^t - v^t\|^2\right] + 2\mathbb{E}[\|v^t\|^2]. \tag{8.18}$$

The first term can be bounded in the following manner:

$$\begin{aligned}
\mathbb{E}\left[\|u^t - v^t\|^2\right] &\leq \mathbb{E}\Big[\Big\|(\nabla f_{i_t}(x^t) - \nabla f_{i_t}(x^{D(t)})) - (\nabla f_{i_t}(\alpha_{i_t}^{D(t)}) - \nabla f_{i_t}(\alpha_{i_t}^t)) \\
&\qquad + \frac{1}{n}\sum_i (\nabla f_i(\alpha_i^t) - \nabla f_i(\alpha_i^{D(t)}))\Big\|^2\Big] \\
&\leq 3\mathbb{E}\left[\left\|\nabla f_{i_t}(x^t) - \nabla f_{i_t}(x^{D(t)})\right\|^2\right] + 3\mathbb{E}\left[\left\|\nabla f_{i_t}(\alpha_{i_t}^{D(t)}) - \nabla f_{i_t}(\alpha_{i_t}^t)\right\|^2\right] \\
&\qquad + 3\mathbb{E}\left[\left\|\frac{1}{n}\sum_i (\nabla f_i(\alpha_i^t) - \nabla f_i(\alpha_i^{D(t)}))\right\|^2\right] \\
&\leq 3\mathbb{E}\left[\left\|\nabla f_{i_t}(x^t) - \nabla f_{i_t}(x^{D(t)})\right\|^2\right] + 3\mathbb{E}\left[\left\|\nabla f_{i_t}(\alpha_{i_t}^{D(t)}) - \nabla f_{i_t}(\alpha_{i_t}^t)\right\|^2\right] \\
&\qquad + \frac{3}{n}\sum_i \mathbb{E}\left[\left\|\nabla f_i(\alpha_i^t) - \nabla f_i(\alpha_i^{D(t)})\right\|^2\right].
\end{aligned} \tag{8.19}$$

The second step follows from Lemma 8.8.3 for $r = 3$. The last step follows from simple application of Jensen's inequality. The first term can be bounded easily in the following manner:

$$\begin{aligned}
\mathbb{E}\left[\|\nabla f_{i_t}(x^t) - \nabla f_{i_t}(x^{D(t)})\|^2\right] &\leq L^2\tau \sum_{d=D(t)}^{t-1} \mathbb{E}\left[\|x^{d+1} - x^d\|_{i_t}^2\right] \\
&\leq L^2\Delta\eta^2\tau \sum_{d=D(t)}^{t-1} \mathbb{E}\left[\|u^d\|^2\right].
\end{aligned}$$

The second and third terms need more delicate analysis. The key insight for our analysis is that at most τ different α_i's differ from time step $D(t)$ to t. This is due to the fact that the delay is bounded by τ and at most one α_i changes at each iteration. Furthermore, whenever there is a change in α_i, it changes to one of the iterates x^j for some $j = \{t - \tau, \ldots, t\}$. With this intuition we bound the second term in the following fashion.

$$
\begin{aligned}
\mathbb{E}\left[\left\|\nabla f_{i_t}(\alpha_i^{D(t)}) - \nabla f_{i_t}(\alpha_{i_t}^t)\right\|^2\right] &\leq \frac{1}{n}\sum_{j=D(t)}^{t-1}\sum_{i\in S}\mathbb{E}\left[\mathbb{1}(i=i_j)\left\|\nabla f_i(x^j) - \nabla f_i(\alpha_i^{D(t)})\right\|^2\right] \\
&\leq \frac{2}{n}\sum_{j=D(t)}^{t-1}\sum_{i\in S}\mathbb{E}\left[\mathbb{1}(i=i_j)\left(\left\|\nabla f_i(x^j) - \nabla f_i(x^*)\right\|^2 + \left\|\nabla f_i(\alpha_i^{D(t)}) - \nabla f_i(x^*)\right\|^2\right)\right] \\
&\leq \frac{2}{n^2}\sum_{j=D(t)}^{t-1}\sum_{i\in S}\mathbb{E}\left[\left\|\nabla f_i(x^j) - \nabla f_i(x^*)\right\|^2\right] + \frac{2}{n^2}\sum_{j=D(t)}^{t-1}\sum_{i\in S}\mathbb{E}\left[\left\|\nabla f_i(\alpha_i^{D(t)}) - \nabla f_i(x^*)\right\|^2\right] \\
&\leq \frac{4L}{n}\sum_{j=D(t)}^{t-1}\mathbb{E}\left[\frac{1}{n}\sum_{i\in S} f_i(x^j) - f_i(x^*) - \langle\nabla f_i(x^*), x^j - x^*\rangle)\right] \\
&\qquad + \frac{4L\tau}{n}\mathbb{E}\left[\frac{1}{n}\sum_{i\in S} f_i(\alpha_i^{D(t)}) - f_i(x^*) - \langle\nabla f_i(x^*), \alpha_i^{D(t)} - x^*\rangle\right].
\end{aligned}
$$

The second inequality follows from Lemma 8.8.3 for $r = 2$. The last step directly follows from Lemma 8.8.1. Note that sum is over indices in S since α_i's for $i \notin S$ do not change during the epoch.

The third term in (8.19) can be bounded by exactly the same technique we used for the second term. The bound, in fact, turns out to identical to the second term since i_t is chosen uniformly at random. Combining all the terms we have

$$
T_6 \leq 2\mathbb{E}[\|v^t\|^2] + 6L^2\Delta\eta^2\tau\sum_{d=D(t)}^{t-1}\mathbb{E}\left[\|u^d\|^2\right] + \frac{48L}{n}\sum_{j=D(t)}^{t-1}\mathbb{E}\left[D_g(x^j, x^*)\right] + \frac{48L\tau}{n}\mathbb{E}\left[G_{D(t)}\right].
$$

The term T_7 can be bounded in a manner similar to one in Theorem 8.3.1 to obtain the following (see proof of Theorem 8.3.1 for details):

$$
\mathbb{E}\langle x^t - x^*, u^t\rangle \leq \mathbb{E}\left[f(x^*) - f(x^t) + \frac{L\Delta\tau\eta^2}{2}\sum_{d=D(t)}^{t-1}\|u^d\|^2\right]. \tag{8.20}
$$

We need the following bound for our analysis:

$$
\begin{aligned}
\sum_{j=0}^{m-1}\left(1-\frac{1}{\kappa}\right)^{m-1-j}\mathbb{E}[\|u^{km+j}\|^2] \leq 2\sum_{j=0}^{m-1}\left(1-\frac{1}{\kappa}\right)^{m-1-j}&\mathbb{E}[\|v^{km+j}\|^2] \\
&+ \sum_{t=km}^{km+m-1} 6L^2\Delta\eta^2\tau\sum_{d=D(t)}^{t-1}\mathbb{E}\left[\|u^d\|^2\right] \\
&+ \sum_{t=km}^{km+m-1}\frac{48L}{n}\sum_{j=D(t)}^{t-1}\mathbb{E}\left[D_g(x^j, x^*)\right] \\
&+ \sum_{t=km}^{km+m-1}\frac{48L\tau}{n}\mathbb{E}\left[G_{D(t)}\right].
\end{aligned}
$$

The above inequality follows directly from the bound on T_6.

Under the condition

$$\eta^2 \leq \left(1 - \frac{1}{\kappa}\right)^{m-1} \frac{1}{12L^2\Delta\tau^2},$$

we have the inequality

$$\begin{aligned}\sum_{j=0}^{m-1}\left(1-\frac{1}{\kappa}\right)^{m-1-j}\mathbb{E}[\|u^{km+j}\|^2] \leq 4\sum_{j=0}^{m-1}&\left(1-\frac{1}{\kappa}\right)^{m-1-j}\mathbb{E}[\|v^{km+j}\|^2] \\ &+ \sum_{t=km}^{km+m-1}\frac{96L}{n}\sum_{j=D(t)}^{t-1}\mathbb{E}\left[D_g(x^j,x^*)\right] \\ &+ \sum_{t=km}^{km+m-1}\frac{96L\tau}{n}\mathbb{E}\left[G_{D(t)}\right]. \qquad (8.21)\end{aligned}$$

This follows from that fact that

$$\sum_{t=km}^{km+m-1} 6L^2\Delta\eta^2\tau \sum_{d=D(t)}^{t-1}\mathbb{E}\left[\|u^d\|^2\right] \leq \frac{1}{2}\sum_{j=0}^{m-1}\left(1-\frac{1}{\kappa}\right)^{m-1-j}\mathbb{E}[\|u^{km+j}\|^2].$$

The above relationship is due to the condition on η and the fact that $d \in [t, D(t)]$ for at most τ values of t. We have the following:

$$\begin{aligned}R_{t+1} &= c\mathbb{E}\left[\|x^t - x^*\|^2\right] + c\eta^2\mathbb{E}\left[\|u^t\|^2\right] + 2c\eta\mathbb{E}\left[\langle x^t - x^*, u^t\rangle\right] + \mathbb{E}\left[G_{t+1}\right] \\ &:= \left(1-\frac{1}{\kappa}\right)R_t + e_t. \qquad (8.22)\end{aligned}$$

We bound e_t in the following manner:

$$\begin{aligned}e_t &= \frac{c}{\kappa}\|x^t - x^*\|^2 + \left(\frac{1}{\kappa}-1\right)\mathbb{E}[G_t] + c\eta^2\mathbb{E}\left[\|u^t\|^2\right] + 2c\eta\mathbb{E}\left[\langle x^t - x^*, u^t\rangle\right] + \mathbb{E}\left[G_{t+1}\right] \\ &= \frac{c}{\kappa}\|x^t - x^*\|^2 + \left(\frac{1}{\kappa}-\frac{1}{n}\right)\mathbb{E}[G_t] + c\eta^2\mathbb{E}\left[\|u^t\|^2\right] + 2c\eta\mathbb{E}\left[\langle x^t - x^*, u^t\rangle\right] + \frac{1}{n}\mathbb{E}[D_g(x^t,x^*)] \\ &\leq -\left(2c\eta - \frac{2c}{\kappa\lambda}\right)\mathbb{E}\left[f(x^t) - f(x^*)\right] + \left(\frac{1}{\kappa}-\frac{1}{n}\right)\mathbb{E}[G_t] + c\eta^2\mathbb{E}[\|u^t\|^2] \\ &\qquad + cL\Delta\tau\eta^3\sum_{d=D(t)}^{t-1}\mathbb{E}\left[\|u^d\|^2\right] + \frac{1}{n}\mathbb{E}[D_g(x^t,x^*)].\end{aligned}$$

The second equality follows from the definition of G_{t+1} (see Equation (8.5)).

$$\mathbb{E}[G_{t+1}] = \left(1-\frac{1}{n}\right)\mathbb{E}\left[G_t\right] + \frac{1}{n}\mathbb{E}[D_g(x^t,x^*)].$$

Applying the recurrence relationship in Equation (8.22) with the derived bound on e_t, we have

$$\begin{aligned} R_{km+m} &\leq \left(1 - \frac{1}{\kappa}\right)^m \tilde{R}_k + \sum_{j=0}^{m-1} \left(1 - \frac{1}{\kappa}\right)^{m-1-j} e_{km+j} \\ &\leq \left(1 - \frac{1}{\kappa}\right)^m \tilde{R}_k + \sum_{j=0}^{m-1} \left(1 - \frac{1}{\kappa}\right)^{m-1-j} e'_{km+j}, \end{aligned}$$

where e'_t is defined as follows

$$\begin{aligned} \tilde{R}_k &= \mathbb{E}\left[c\|\tilde{x}^k - x^*\|^2 + \tilde{G}_k\right] \\ e'_t &= -\left(2c\eta - \frac{2c}{\kappa\lambda}\right)\mathbb{E}\left[f(x^t) - f(x^*)\right] + \left(\frac{1}{\kappa} - \frac{1}{n}\right)\mathbb{E}[G_t] \\ &\quad + \left(c\eta^2 + \left(1 - \frac{1}{\kappa}\right)^{-\tau} cL\Delta\tau^2\eta^3\right)\mathbb{E}[\|u^t\|^2] + \frac{1}{n}\mathbb{E}[D_g(x^t, x^*)]. \end{aligned}$$

We use the following notation for ease of exposition:

$$\zeta = \left(c\eta^2 + \left(1 - \frac{1}{\kappa}\right)^{-\tau} cL\Delta\tau^2\eta^3\right).$$

This last inequality follows from that fact that the delay is at most τ. In particular, each index $j \in \{D(t) \ldots, t\}$ for at most τ times. Substituting the bound in Equation (8.21), we get the following:

$$\begin{aligned} R_{km+m} \leq &\left(1 - \frac{1}{\kappa}\right)^m \tilde{R}_k - \left(2c\eta - \frac{2c}{\kappa\lambda}\right)\sum_{j=0}^{m-1}\left(1 - \frac{1}{\kappa}\right)^{m-1-j}\mathbb{E}\left[f(x^{km+j}) - f(x^*)\right] \\ &+ 4\zeta\sum_{j=0}^{m-1}\left(1 - \frac{1}{\kappa}\right)^{m-1-j}\mathbb{E}[\|v^{km+j}\|^2] \\ &+ \left[\frac{96\zeta L\tau}{n}\left(1 - \frac{1}{\kappa}\right)^{-\tau} + \frac{1}{n}\right]\sum_{j=0}^{m-1}\left(1 - \frac{1}{\kappa}\right)^{m-1-j}\mathbb{E}\left[D_g(x^{km+j}, x^*)\right] \\ &+ \left[\frac{1}{\kappa} + \frac{96\zeta L\tau}{n}\left(1 - \frac{1}{\kappa}\right)^{-\tau} - \frac{1}{n}\right]\sum_{j=0}^{m-1}\left(1 - \frac{1}{\kappa}\right)^{m-1-j}\mathbb{E}\left[G_{D(km+j)}\right]. \quad (8.23) \end{aligned}$$

We now use the following previously used bound on v^t (see bound T_2 in the proof of Theorem 8.2.1).

$$\mathbb{E}[\|v^t\|^2] \leq 2L\left(1 + \frac{1}{\beta}\right)\left[G_t + D_h(\tilde{x}^k, x^*)\right] + 2L(1+\beta)\mathbb{E}[f(x^t) - f(x^*)].$$

Substituting the above bound on v^t in Equation (8.23), we get the following:

$$\begin{aligned}
R_{km+m} \leq & \left(1-\frac{1}{\kappa}\right)^m \tilde{R}_k - \left[2c\eta - 8\zeta L(1+\beta) - \frac{2c}{\kappa\lambda} - \frac{96\zeta L\tau}{n}\left(1-\frac{1}{\kappa}\right)^{-\tau} - \frac{1}{n}\right] \times \\
& \qquad \sum_{j=0}^{m-1}\left(1-\frac{1}{\kappa}\right)^{m-1-j} \mathbb{E}\left[f(x^{km+j}) - f(x^*)\right] \\
& + \left[\frac{1}{\kappa} + 8\zeta L\left(1+\frac{1}{\beta}\right) + \frac{96\zeta L\tau}{n}\left(1-\frac{1}{\kappa}\right)^{-\tau} - \frac{1}{n}\right] \times \\
& \qquad \sum_{j=0}^{m-1}\left(1-\frac{1}{\kappa}\right)^{m-1-j} \mathbb{E}\left[G_{km+j}\right] \\
& + 8\zeta L\left(1+\frac{1}{\beta}\right)\sum_{j=0}^{m-1}\left(1-\frac{1}{\kappa}\right)^{m-1-j} \mathbb{E}\left[D_h(\tilde{x}^k, x^*)\right] \\
\leq & \frac{2c}{\lambda}\left(1-\frac{1}{\kappa}\right)^m \mathbb{E}\left[f(\tilde{x}^k) - f(x^*)\right] + \left(1-\frac{1}{\kappa}\right)^m \mathbb{E}\left[\tilde{G}_k\right] \\
& - \left[2c\eta - 8\zeta L(1+\beta) - \frac{2c}{\kappa\lambda} - \frac{96\zeta L\tau}{n}\left(1-\frac{1}{\kappa}\right)^{-\tau} - \frac{1}{n}\right] \times \\
& \qquad \sum_{j=0}^{m-1}\left(1-\frac{1}{\kappa}\right)^{m-1-j} \mathbb{E}\left[f(x^{km+j}) - f(x^*)\right] \\
& + 8\zeta L\left(1+\frac{1}{\beta}\right)\kappa\left[1-\left(1-\frac{1}{\kappa}\right)^m\right] \mathbb{E}\left[D_h(\tilde{x}^k, x^*)\right]. \qquad (8.24)
\end{aligned}$$

The first step is due to the Bregman divergence based inequality $D_f(x, x^*) \geq D_g(x, x^*)$. The second step follows from the expanding $\tilde{R}_k$ and using the strong convexity of function f. We now use the fact that $\tilde{x}^{k+1}$ is chosen randomly from $\{x^{km}, \dots, x^{km+m-1}\}$ with probabilities proportional to $\{(1-1/\kappa)^{m-1}, \dots, 1\}$ we have the following consequence of the above inequality. We use the following notation:

$$\begin{aligned}
\gamma_a &= \kappa\left[1-\left(1-\frac{1}{\kappa}\right)^m\right]\left[2c\eta - 8\zeta L(1+\beta) - \frac{2c}{\kappa\lambda} - \frac{96\zeta L\tau}{n}\left(1-\frac{1}{\kappa}\right)^{-\tau} - \frac{1}{n}\right] \\
\theta_a &= \max\left\{\left[\frac{2c}{\gamma_a\lambda}\left(1-\frac{1}{\kappa}\right)^m + \frac{8\zeta L\left(1+\frac{1}{\beta}\right)}{\gamma_a}\kappa\left[1-\left(1-\frac{1}{\kappa}\right)^m\right]\right], \left(1-\frac{1}{\kappa}\right)^m\right\}.
\end{aligned}$$

Using the above notation, we have the following inequality from Equation (8.7).

$$\mathbb{E}\left[f(\tilde{x}^{k+1}) - f(x^*) + \frac{1}{\gamma_a}\tilde{G}_{k+1}\right] \leq \theta_a\, \mathbb{E}\left[f(\tilde{x}^k) - f(x^*) + \frac{1}{\gamma_a}\tilde{G}_k\right].$$

where $\theta < 1$ is a constant that depends on the parameters used in the algorithm.

□

8.7 Proof of Theorem 8.4.1

Proof. Using exactly the same argument as in first part of Theorem 8.2.1, we have the following:

$$R_{t+1} \leq R_t + c\eta^2 \mathbb{E}\left[\|v^t\|^2\right] - 2c\eta \mathbb{E}\left[f(x^t) - f(x^*)\right] - \frac{1}{n}\mathbb{E}[G_t] + \frac{1}{n}\mathbb{E}[D_g(x^t, x^*)] := R_t + b'_t.$$

The terms $\mathbb{E}\left[\|v^t\|^2\right]$ can be bounded in the following fashion:

$$\begin{aligned}
\mathbb{E}\left[\|v^t\|^2\right] &\leq \left(1 + \frac{1}{\beta}\right) \mathbb{E}\left[\|\nabla f_{i_t}(\alpha^t_{i_t}) - \nabla f_{i_t}(x^*)\|^2\right] + (1+\beta)\mathbb{E}\left[\|\nabla f_{i_t}(x^t) - \nabla f_{i_t}(x^*)\|^2\right] \\
&\leq \frac{2L}{n}\left(1 + \frac{1}{\beta}\right) \mathbb{E}\sum_i \left[f_i(\alpha^t_i) - f(x^*) - \langle \nabla f_i(x^*), \alpha^t_i - x^* \rangle\right] \\
&\qquad + \frac{2L}{n}(1+\beta)\mathbb{E}\sum_i \left[f_i(x^t) - f(x^*)\right] \\
&\leq 2L\left(1 + \frac{1}{\beta}\right)\left[G_t + D_h(\tilde{x}^k, x^*)\right] + 2L(1+\beta)\mathbb{E}[f(x^t) - f(x^*)].
\end{aligned}$$

The first inequality and second inequalities directly follow from Lemma 3 of [33] and simple application of Lemma 8.8.1 respectively. The third inequality follows from the definition of G_t and the fact that $\alpha^t_i = \tilde{x}^k$ for all $i \notin S$ and $t \in \{km, \ldots, km+m\}$.

Substituting the above bound in b'_t, we get

$$\begin{aligned}
b'_t &\leq -\left[2c\eta - 2cL\eta^2(1+\beta)\right]\mathbb{E}\left[f(x^t) - f(x^*)\right] \\
&\quad + \left(2cL\eta^2\left(1 + \frac{1}{\beta}\right) - \frac{1}{n}\right)\mathbb{E}[G_t] + \frac{1}{n}\mathbb{E}[D_g(x^t, x^*)] \\
&\quad + 2cL\eta^2\left(1 + \frac{1}{\beta}\right)\mathbb{E}\left[D_h(\tilde{x}^k, x^*)\right] \\
&\leq -\left[2c\eta - 2cL\eta^2(1+\beta) - \frac{1}{n}\right]\mathbb{E}\left[f(x^t) - f(x^*)\right] \\
&\quad + \left(2cL\eta^2\left(1 + \frac{1}{\beta}\right) - \frac{1}{n}\right)\mathbb{E}[G_t] + 2cL\eta^2\left(1 + \frac{1}{\beta}\right)\mathbb{E}\left[D_h(\tilde{x}^k, x^*)\right] \\
&\leq -\left[2c\eta - 2cL\eta^2(1+\beta) - \frac{1}{n}\right]\mathbb{E}\left[f(x^t) - f(x^*)\right] + 2cL\eta^2\left(1 + \frac{1}{\beta}\right)\mathbb{E}\left[D_h(\tilde{x}^k, x^*)\right]
\end{aligned} \tag{8.25}$$

The second inequality follows from Lemma 8.8.2. In particular, we use the fact that $f(x) - f(x^*) = D_f(x, x^*)$ and $D_f(x, x^*) = D_g(x, x^*) + D_h(x, x^*) \geq D_g(x, x^*)$. The third inequality follows from the following for the choice of our parameters:

$$2Lc\eta^2\left(1 + \frac{1}{\beta}\right) \leq \frac{1}{n}.$$

Applying the recursive relationship on R_{t+1} for m iterations, we get

$$R_{km+m} \leq R_{km} + \sum_{j=0}^{m-1} b'_{km+j}.$$

Substituting the bound on b'_t from Equation (8.6) in the above equation we get the following inequality:

$$\begin{aligned} R_{km+m} \leq R_{km} & \\ & - \sum_{j=0}^{m-1} \left(2c\eta(1 - L\eta(1+\beta)) - \frac{1}{n}\right) \mathbb{E}\left[f(x^{km+j}) - f(x^*)\right] \\ & + \sum_{j=0}^{m-1} 2Lc\eta^2 \left(1 + \frac{1}{\beta}\right) \mathbb{E}\left[D_h(\tilde{x}^k, x^*)\right]. \end{aligned}$$

We define the following quantity:

$$P_k = R_{km} + 2mLc\eta^2 \left(1 + \frac{1}{\beta}\right) \mathbb{E}\left[D_h(\tilde{x}^k, x^*)\right]. \tag{8.26}$$

Using this definition, we have

$$\left(2c\eta(1 - L\eta(1+\beta)) - \frac{1}{n} - 2Lc\eta^2\left(1 + \frac{1}{\beta}\right)\right) \sum_{j=0}^{m-1} \mathbb{E}\left[f(x^{km+j}) - f(x^*)\right] \leq P_k - P_{k+1}.$$

This directly follows from the definition of P_k, $\tilde{x}^k$ and the fact $D_h(x, x^*) \leq f(x) - f(x^*)$. Now using the telescopy sum, we have the main result:

$$\mathbb{E}\left[f(x_a^T) - f(x^*)\right] \leq \frac{P_0}{T\left(2c\eta(1 - L\eta(1+\beta)) - \frac{1}{n} - 2Lc\eta^2\left(1 + \frac{1}{\beta}\right)\right)}.$$

□

8.8 Proof of Theorem 8.4.2

Proof. Let the present epoch be $k+1$. For simplicity, we assume that the iterates x and A used in the each iteration are from the same time step (index) i.e., $D(t) = D'(t)$ for all $t \in T$. Recall that $D(t)$ and $D'(t)$ denote the index used in the t^{th} iteration of the algorithm. Our analysis can be extended to the case of $D(t) \neq D'(t)$ in a straightforward manner. We expand function f as $f(x) = g(x) + h(x)$ where $g(x) = \frac{1}{n}\sum_{i \in S} f_i(x)$ and

$h(x) = \frac{1}{n}\sum_{i \notin S} f_i(x)$. We define the following:

$$u^t = \frac{1}{\eta}(x^{t+1} - x^t) = -\left[\nabla f_{i_t}(x^{D(t)}) - \nabla f_{i_t}(\alpha_{i_t}^{D(t)}) + \frac{1}{n}\sum_i \nabla f_i(\alpha_i^{D(t)})\right]$$

$$v^t = -\left[\nabla f_{i_t}(x^t) - \nabla f_{i_t}(\alpha_{i_t}^t) + \frac{1}{n}\sum_i \nabla f_i(\alpha_i^t)\right].$$

We use the same Lyapunov function used in Theorem 8.2.1. We recall the following definitions:

$$G_t = \frac{1}{n}\sum_{i \in S}\left(f_i(\alpha_i^t) - f_i(x^*) - \langle \nabla f_i(x^*), \alpha_i^t - x^*\rangle\right)$$

$$R_t = \mathbb{E}\left[c\|x^t - x^*\|^2 + G_t\right].$$

Using unbiasedness of the gradient we have $\mathbb{E}[u^t] = -\nabla f(x^{D(t)})$ and $\mathbb{E}[v^t] = -\nabla f(x^t)$. Using this observation, we have the following:

$$\begin{aligned} c\mathbb{E}[\|x^{t+1} - x^*\|^2] &= c\mathbb{E}[\|x^t + \eta u^t - x^*\|^2] \\ &= c\mathbb{E}\left[\|x^t - x^*\|^2\right] + c\eta^2 \underbrace{\mathbb{E}\left[\|u^t\|^2\right]}_{T_6} + 2c\eta \underbrace{\mathbb{E}\left[\langle x^t - x^*, u^t\rangle\right]}_{T_7}. \end{aligned} \tag{8.27}$$

Using the argument in Theorem 8.3.2, we obtain the following bounds on T_6 and T_7:

$$T_6 \leq 2\mathbb{E}[\|v^t\|^2] + 6L^2\Delta\eta^2\tau \sum_{d=D(t)}^{t-1} \mathbb{E}\left[\|u^d\|^2\right] + \frac{48L}{n}\sum_{j=D(t)}^{t-1}\mathbb{E}\left[D_g(x^j, x^*)\right] + \frac{48L\tau}{n}\mathbb{E}\left[G_{D(t)}\right],$$

$$T_7 \leq \mathbb{E}\left[f(x^*) - f(x^t) + \frac{L\Delta\tau\eta^2}{2}\sum_{d=D(t)}^{t-1}\|u^d\|^2\right].$$

We need the following bound for our analysis:

$$\begin{aligned} \sum_{j=0}^{m-1}\mathbb{E}[\|u^{km+j}\|^2] \leq 2\sum_{j=0}^{m-1}\mathbb{E}[\|v^{km+j}\|^2] &+ \sum_{t=km}^{km+m-1} 6L^2\Delta\eta^2\tau \sum_{d=D(t)}^{t-1}\mathbb{E}\left[\|u^d\|^2\right] \\ &+ \sum_{t=km}^{km+m-1}\frac{48L}{n}\sum_{j=D(t)}^{t-1}\mathbb{E}\left[D_g(x^j, x^*)\right] \\ &+ \sum_{t=km}^{km+m-1}\frac{48L\tau}{n}\mathbb{E}\left[G_{D(t)}\right]. \end{aligned}$$

The above inequality follows directly from the bound on T_6. Under the condition $\eta^2 \leq \frac{1}{12L^2\Delta\tau^2}$, we have the following inequality:

$$\begin{aligned}\sum_{j=0}^{m-1} \mathbb{E}[\|u^{km+j}\|^2] \leq 4 \sum_{j=0}^{m-1} \mathbb{E}[\|v^{km+j}\|^2] \\ + \sum_{t=km}^{km+m-1} \frac{96L}{n} \sum_{j=D(t)}^{t-1} \mathbb{E}\left[D_g(x^j, x^*)\right] \\ + \sum_{t=km}^{km+m-1} \frac{96L\tau}{n} \mathbb{E}\left[G_{D(t)}\right].\end{aligned} \tag{8.28}$$

This follows from that fact that

$$\sum_{t=km}^{km+m-1} 6L^2\Delta\eta^2\tau \sum_{d=D(t)}^{t-1} \mathbb{E}\left[\|u^d\|^2\right] \leq \frac{1}{2} \sum_{j=0}^{m-1} \mathbb{E}[\|u^{km+j}\|^2]$$

due to the condition mentioned above. We have the following:

$$\begin{aligned}R_{t+1} &= c\mathbb{E}\left[\|x^t - x^*\|^2\right] + c\eta^2\mathbb{E}\left[\|u^t\|^2\right] + 2c\eta\mathbb{E}\left[\langle x^t - x^*, u^t\rangle\right] + \mathbb{E}\left[G_{t+1}\right] \\ &:= R_t + e'_t.\end{aligned} \tag{8.29}$$

We bound e'_t in the following manner:

$$\begin{aligned}e'_t &= c\eta^2\mathbb{E}\left[\|u^t\|^2\right] - \mathbb{E}\left[G_t\right] + 2c\eta\mathbb{E}\left[\langle x^t - x^*, u^t\rangle\right] + \mathbb{E}\left[G_{t+1}\right] \\ &= -\frac{1}{n}\mathbb{E}[G_t] + c\eta^2\mathbb{E}\left[\|u^t\|^2\right] + 2c\eta\mathbb{E}\left[\langle x^t - x^*, u^t\rangle\right] + \frac{1}{n}\mathbb{E}[D_g(x^t, x^*)] \\ &\leq -\frac{1}{n}\mathbb{E}[G_t] - 2c\eta\mathbb{E}\left[f(x^t) - f(x^*)\right] + c\eta^2\mathbb{E}[\|u^t\|^2] \\ &\qquad + cL\Delta\tau\eta^3 \sum_{d=D(t)}^{t-1} \mathbb{E}\left[\|u^d\|^2\right] + \frac{1}{n}\mathbb{E}[D_g(x^t, x^*)] \\ &\leq -\frac{1}{n}\mathbb{E}[G_t] - \left(2c\eta - \frac{1}{n}\right)\mathbb{E}\left[f(x^t) - f(x^*)\right] + c\eta^2\mathbb{E}[\|u^t\|^2] + cL\Delta\tau\eta^3 \sum_{d=D(t)}^{t-1} \mathbb{E}\left[\|u^d\|^2\right]\end{aligned}$$

The second equality follows from the definition of G_{t+1} (see Equation (8.5)).

$$\mathbb{E}[G_{t+1}] = \left(1 - \frac{1}{n}\right)\mathbb{E}\left[G_t\right] + \frac{1}{n}\mathbb{E}[D_g(x^t, x^*)].$$

The third and fourth inequalities follow from the fact that $G_t \geq 0$ and $D_g(x^t, x^*) \leq f(x^t) - f(x^*)$ respectively. Applying the recurrence relationship in Equation (8.29) with the derived bound on e'_t, we have

$$R_{km+m} \leq R_{km} + \sum_{j=0}^{m-1} e'_{km+j} \leq R_{km} - \left(2c\eta - \frac{1}{n}\right) \sum_{j=0}^{m-1} \mathbb{E}\left[f(x^{km+j}) - f(x^*)\right] + \left(c\eta^2 + cL\Delta\tau^2\eta^3\right) \sum_{j=0}^{m-1} \mathbb{E}[\|u^{km+j}\|^2] - \frac{1}{n}\sum_{j=0}^{m-1} \mathbb{E}[G_{km+j}]$$

This last inequality follows from that fact that the delay is at most τ. In particular, each index $j \in \{D(t)\dots,t\}$ for at most τ times. Substituting the bound in Equation (8.28), we get the following:

$$\begin{aligned} R_{km+m} \leq R_{km} &- \left(2c\eta - \frac{1}{n}\right) \sum_{j=0}^{m-1} \mathbb{E}\left[f(x^{km+j}) - f(x^*)\right] \\ &+ 4\left(c\eta^2 + cL\Delta\tau^2\eta^3\right) \sum_{j=0}^{m-1} \mathbb{E}[\|v^{km+j}\|^2] - \frac{1}{n}\sum_{j=0}^{m-1} \mathbb{E}[G_{km+j}] \\ &+ \frac{96L\left(c\eta^2 + cL\Delta\tau^2\eta^3\right)}{n} \sum_{t=km}^{km+m-1} \sum_{j=D(t)}^{t-1} \mathbb{E}\left[D_g(x^j, x^*)\right] \\ &+ \frac{96L\tau\left(c\eta^2 + cL\Delta\tau^2\eta^3\right)}{n} \sum_{t=km}^{km+m-1} \mathbb{E}\left[G_{D(t)}\right]. \end{aligned} \tag{8.30}$$

We now use the following previously used bound on v^t (see bound T_2 in the proof of Theorem 8.2.1).

$$\mathbb{E}[\|v^t\|^2] \leq 2L\left(1 + \frac{1}{\beta}\right)\left[G_t + D_h(\tilde{x}^k, x^*)\right] + 2L(1+\beta)\mathbb{E}[f(x^t) - f(x^*)].$$

We use ζ' to denote $c\eta^2 + cL\Delta\tau^2\eta^3$. Substituting the above bound on v^t in Equation (8.30), we get the following:

$$\begin{aligned} R_{km+m} &\leq R_{km} - \left[2c\eta - 8\zeta' L(1+\beta) - \frac{96\zeta' L\tau}{n} - \frac{1}{n}\right] \sum_{j=0}^{m-1} \mathbb{E}\left[f(x^{km+j}) - f(x^*)\right] \\ &\quad + \left[8\zeta' L\left(1 + \frac{1}{\beta}\right) + \frac{96\zeta' L\tau}{n} - \frac{1}{n}\right] \sum_{j=0}^{m-1} \mathbb{E}\left[G_{km+j}\right] \\ &\quad + 8\zeta' L\left(1 + \frac{1}{\beta}\right) \sum_{j=0}^{m-1} \mathbb{E}\left[D_h(\tilde{x}^k, x^*)\right] \\ &\leq R_{km} - \left[2c\eta - 8\zeta' L(1+\beta) - \frac{96\zeta' L\tau}{n} - \frac{1}{n}\right] \sum_{j=0}^{m-1} \mathbb{E}\left[f(x^{km+j}) - f(x^*)\right] \\ &\quad + 8m\zeta' L\left(1 + \frac{1}{\beta}\right) \mathbb{E}\left[D_h(\tilde{x}^k, x^*)\right]. \end{aligned} \tag{8.31}$$

We define the following quantity:

$$P'_k = R_{km} + 8m\zeta' L \left(1 + \frac{1}{\beta}\right) \mathbb{E}\left[D_h(\tilde{x}^k, x^*)\right]. \tag{8.32}$$

Using this definition, we have

$$\left[2c\eta - 8\zeta' L(1+\beta) - \frac{96\zeta' L\tau}{n} - \frac{1}{n} - 8\zeta' L\left(1 + \frac{1}{\beta}\right)\right] \sum_{j=0}^{m-1} \mathbb{E}\left[f(x^{km+j}) - f(x^*)\right] \leq P'_k - P'_{k+1}.$$

This directly follows from the definition of P'_k, $\tilde{x}^k$ and the fact $D_h(x, x^*) \leq f(x) - f(x^*)$. Now using the telescopy sum, we have the main result:

$$\mathbb{E}\left[f(x_a^T) - f(x^*)\right] \leq \frac{P'_0}{T\left[2c\eta - 8\zeta' L(1+\beta) - \frac{96\zeta' L\tau}{n} - \frac{1}{n} - 8\zeta' L\left(1 + \frac{1}{\beta}\right)\right]}.$$

□

Other Lemmatta

Lemma 8.8.1. *[71] For any $\alpha_i \in \mathbb{R}^d$ where $i \in [n]$ and x^*, we have*

$$\mathbb{E}\left[\|\nabla f_{i_t}(\alpha_{i_t}) - \nabla f_{i_t}(x^*)\|^2\right] \leq \frac{2L}{n} \sum_i \left[f_i(\alpha_i) - f(x^*) - \langle \nabla f_i(x^*), \alpha_i - x^* \rangle\right].$$

Lemma 8.8.2. *Suppose $f : \mathbb{R}^d \to \mathbb{R}$ and $f = g + h$ where f, g and h are convex and differentiable. x^* is the optimal solution to $\arg\min_x f(x)$ then we have the following*

$$\begin{aligned} D_f(x, x^*) &= f(x) - f(x^*) \\ D_f(x, x^*) &= D_g(x, x^*) + D_h(x, x^*) \\ D_f(x, x^*) &\geq D_g(x, x^*). \end{aligned}$$

Proof. The proof follows trivially from the fact that x^* is the optimal solution and linearity and non-negative properties of Bregman divergence. □

Lemma 8.8.3. *For random variables $z_1, \ldots, z_r$, we have*

$$\mathbb{E}\left[\|z_1 + ... + z_r\|^2\right] \leq r\mathbb{E}\left[\|z_1\|^2 + ... + \|z_r\|^2\right].$$

Chapter 9

Asynchronous Randomized Coordinate Descent Algorithms for ERM

9.1 Introduction

In this chapter, we develop, analyze, and implement asynchronous randomized coordinate descent methods for the following composite objective convex problem with *non-separable* linear constraints

$$\min_x F(x) := f(x) + h(x) \quad \text{s.t.} \quad Ax = 0. \tag{9.1}$$

Here, $f : \mathbb{R}^n \to \mathbb{R}$ is assumed to be continuously differentiable and convex, while $h : \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ is lower semi-continuous, convex, coordinate-wise separable, but not necessarily smooth; the linear constraints (LC) are specified by a matrix $A \in \mathbb{R}^{m \times n}$, for which $m \ll n$, and a certain structure (see Section 9.4) is assumed. The reader may wonder the relationship between aforementioned problem and ERM problem of our interest, but observe that any ERM problem can be rewritten in dual form (9.1) as follows:

$$\min_x \frac{1}{n}\sum_{i=1}^n f_i(x) \equiv \min_{\{x_i = x, \forall i \in [n]\}} \frac{1}{n}\sum_{i=1}^n f_i(x_i) \equiv \min_\lambda \frac{1}{n}\sum_{i=1}^n f_i^*(\lambda_i) \quad s.t \ \sum_{i=1}^n \lambda_i = 0.$$

Thus, an asynchronous coordinate descent method for (9.1), consequently, yields an asynchronous dual algorithm for ERM problems. Coordinate descent (CD) methods are conceptually among the simplest schemes for unconstrained optimization—they have been studied for a long time (see e.g., [10, 17, 123]), and are now enjoying greatly renewed interest. Their resurgence is rooted in successful applications in machine learning [63, 64], statistics [41, 65], and many other areas—see [145, 146, 157] and references therein for more examples.

A catalyst to the theoretical as well as practical success of CD methods has been randomization. (The idea of randomized algorithms for optimization methods is of course much older, see e.g., [127].) Indeed, generic non-randomized CD has resisted complexity analysis, though there is promising recent work [62, 151, 171]; remarkably for randomized CD for smooth convex optimization, Nesterov [112, 114] presented an analysis of global iteration complexity. This work triggered several improvements, such as [146, 147], who simplified and extended the analysis to include separable nonsmooth terms. Randomization has also been crucial to a host of other CD algorithms and analyses [21, 64, 91, 106, 128, 145, 147, 157, 158, 166].

Almost all of the aforementioned CD methods assume essentially unconstrained problems, which at best allow separable constraints. In contrast, we develop, analyze, and implement randomized CD methods for the following composite objective convex problem with *non-separable* linear constraints. Problem (9.1) subsumes the usual regularized optimization problems pervasive in machine learning for the simplest ($m = 0$) case. In the presence of linear constraints ($m > 0$), Problem (9.1) assumes a form used in the classic Alternating Direction Method of Multipliers (ADMM) [44, 51]. The principal difference between our approach and ADMM is that the latter treats the entire variable $x \in \mathbb{R}^n$ as a single block, whereas we use the structure of A to split x into b smaller blocks. Familiar special cases of Problem (9.1) include SVM (with bias) dual, fused Lasso and group Lasso [167], and linearly constrained least-squares regression [52, 82].

Recently, Necoara et al. [106] studied a special case of Problem (9.1) that sets $h \equiv 0$ and assumes a single sum constraint. They presented a randomized CD method that starts with a feasible solution and at each iteration updates a pair of coordinates to ensure descent on the objective while maintaining feasibility. This scheme is reminiscent of the well-known SMO procedure for SVM optimization [121]. For smooth convex problems with n variables, Necoara et al. [106] prove an $O(1/\epsilon)$ rate of convergence. More recently, in [105] considered a generalization to the general case $Ax = 0$ (assuming h is coordinatewise separable).

Unfortunately, the analysis in [105] yields an extremely pessimistic complexity result:

Theorem 9.1.1 ([105])**.** *Consider Problem* (9.1) *with h being coordinatewise separable, and $A \in \mathbb{R}^{m\times n}$ with b blocks. Then, the CD algorithm in [105] takes no more than $O(b^m/\epsilon)$ iterations to obtain a solution of ϵ-accuracy.*

This result is exponential in the number of constraints and too severe even for small-scale problems!

We present randomized CD methods, and prove that for important special cases (mainly $h \equiv 0$ *or* A is a sum constraint) we can obtain global iteration complexity that *does not have* an intractable dependence on either the number of coordinate blocks (b), or on the number of linear constraints (m). Previously, Tseng and Yun [169] also studied a linearly coupled block-CD method based on the Gauss-Southwell choice; however, their complexity analysis applies only to the special $m = 0$ and $m = 1$ cases.

To our knowledge, ours is the first work on CD for problems with more than one ($m > 1$) linear constraints that presents such results.

Contributions. In light of the above background, our primary contributions in this chapter are as follows:

- Convergence rate analysis of a randomized block-CD method for the smooth case ($h \equiv 0$) with $m \geq 1$ general linear constraints.

- ◦ A tighter convergence analysis for the composite function optimization ($h \neq 0$) than [105] in the case of sum constraint.
- ◦ An asynchronous CD algorithm for Problem (9.1).
- ◦ A stochastic CD method with convergence analysis for solving problems with a separable loss $f(x) = (1/N)\sum_{i=1}^{N} f_i(x)$.

Additional related work. As noted, CD methods have a long history in optimization and they have gained tremendous recent interest. We cannot hope to do full justice to all the related work, but refer the reader to [146, 147] and [91] for more thorough coverage. Classically, local linear convergence was analyzed in [94]. Global rates for randomized block coordinate descent (BCD) were pioneered by Nesterov [112], and have since then been extended by various authors [13, 146, 147, 171]. The related family of Gauss-Seidel like analyses for ADMM have also recently gained prominence [61]. A combination of randomized block-coordinate ideas with Frank-Wolfe methods was recently presented in [80], though algorithmically the Frank-Wolfe approach is very different as it relies on non projection based oracles.

9.2 Preliminaries

In this section, we further explain our model and assumptions. We assume that the entire space $\mathbb{R}^n$ is decomposed into b *blocks*, i.e., $x = [x_1^\top, \cdots, x_b^\top]^\top$ where $x \in \mathbb{R}^n$, $x_i \in \mathbb{R}^{n_i}$ for all $i \in [b]$, and $n = \sum_i n_i$. For any $x \in \mathbb{R}^n$, we use x_i to denote the i^{th} block of x. We model communication constraints in our algorithms by viewing variables as nodes in a connected graph $G := (V, E)$. Specifically, node $i \in V \equiv [b]$ corresponds to variable x_i, while an edge $(i, j) \in E \subset V \times V$ is present if nodes i and j can exchange information. We use "pair" and "edge" interchangeably.

For a differentiable function f, we use $f_{i_1 \cdots i_p}$ and $\nabla_{i_1 \cdots i_p} f(x)$ (or $\nabla_{x_{i_1} \cdots x_{i_p}} f(x)$) to denote the restriction of the function and its partial gradient to coordinate blocks $(x_{i_1}, \cdots, x_{i_p})$. For any matrix B with n columns, we use B_i to denote the columns of B corresponding to x_i and B_{ij} to denote the columns of B corresponding to x_i and x_j. We use U to denote the $n \times n$ identity matrix and hence U_i is a matrix that places an n_i dimensional vector into the corresponding block of an n dimensional vector.

We make the following standard assumption on the partial gradients of f.

Assumption 1. *The function has block-coordinate Lipschitz continuous gradient, i.e.,*

$$\|\nabla_i f(x) - \nabla_i f(x + U_i h)\| \leq L_i \|h_i\| \textit{ for all } x \in \mathbb{R}^n, .$$

Assumption 1 is similar to the typical Lipschitz continuous gradients assumed in first-order methods and it is necessary to ensure convergence of block-coordinate methods. When functions f_i and f_j have Lipschitz continuous gradients with constants L_i and L_j respectively, one can show that the function f_{ij} has a Lipschitz continuous gradient with $L_{ij} = L_i + L_j$ [105, Lemma 1]. The following result is standard.

Lemma 9.2.1. *For any function $g : \mathbb{R}^n \to \mathbb{R}$ with L-Lipschitz continuous gradient ∇g, we have*

$$g(x) \leq g(y) + \langle \nabla g(y), x - y \rangle + \tfrac{L}{2}\|x-y\|^2 \; x, y \in \mathbb{R}^n.$$

Following [146, 169], we also make the following assumption on the structure of h.

Assumption 2. *The nonsmooth function h is block separable, i.e., $h(x) = \sum_i h_i(x_i)$.*

This assumption is critical to composite optimization using CD methods. We also assume access to an oracle that returns function values and *partial gradients* at any points and iterates of the optimization algorithm.

9.3 Algorithm

We are now ready to present our randomized CD methods for Problem (9.1) in various settings. We first study composite minimization (§9.3.1) and later look at asynchronous (§9.3.2) and stochastic (§9.3.3) variants. The main idea underlying our algorithms is to pick a random pair $(i, j) \in E$ of variables (blocks) at each iteration, and to update them in a manner which maintains feasibility and ensures progress in optimization.

9.3.1 Composite Minimization

We begin with the nonsmooth setting, where $h \not\equiv 0$. We start with a feasible point x^0. Then, at each iteration we pick a random pair $(i, j) \in E$ of variables and minimize the first-order Taylor expansion of the loss f around the current iterate while maintaining feasibility. Formally, this involves performing the update

$$\begin{aligned} Z(f, x, (i,j), \alpha) := \operatorname*{arg\,min}_{A_{ij} d_{ij} = 0} f(x) + \langle \nabla_{ij} f(x), d_{ij} \rangle \\ + (2\alpha)^{-1}\|d_{ij}\|^2 + h(x + U_{ij} d_{ij}), \end{aligned} \tag{9.2}$$

where $\alpha > 0$ is a stepsize parameter and d_{ij} is the update. The right hand side of Equation (9.2) upper bounds f at $x + U_{ij} d_{ij}$, as seen by using Assumption 1 and Lemma 9.2.1. If $h(x) \equiv 0$, minimizing Equation (9.2) yields

$$\begin{aligned} \lambda &\leftarrow \alpha (A_i A_i^\top + A_j A_j^\top)^+ \left(A_i \nabla_i f(x) + A_j \nabla_j f(x) \right) \\ d_i &\leftarrow -\alpha \nabla_i f(x) + A_i^\top \lambda \\ d_j &\leftarrow -\alpha \nabla_j f(x) + A_j^\top \lambda \end{aligned} \tag{9.3}$$

Algorithm 21 presents the resulting method.

Note that since we start with a feasible point x^0 and the update d^k satisfies $A d^k = 0$, the iterate x^k is always feasible. However, it can be shown that a necessary condition for Equation (9.2) to result in a non-zero update is that A_i and A_j span the same column space. If the constraints are not block separable (i.e. for any partitioning of blocks

$x_1, \dots, x_b$ into two groups, there is a constraint that involves blocks from both groups), a typical way to satisfy the aforementioned condition is to require A_i to be full row-rank for all $i \in [b]$. This constraints the minimum block size to be chosen in order to apply randomized CD.

Theorem 9.4.1 describes convergence of Algorithm 21 for the smooth case ($h \equiv 0$), while Theorem 9.4.4 considers the nonsmooth case under a suitable assumption on the structure of the interdependency graph G—both results are presented in Section 9.4.

```
1: $x^0 \in \mathbb{R}^n$ such that $Ax^0 = 0$
2: for $k \geq 0$ do
3:     Select a random edge $(i_k, j_k) \in E$ with probability $p_{i_k j_k}$
4:     $d^k \leftarrow U_{i_k j_k} Z(f, x^k, (i_k, j_k), \alpha_k / L_{i_k j_k})$
5:     $x^{k+1} \leftarrow x^k + d^k$
6:     $k \leftarrow k + 1$
7: end for
```

Algorithm 21: Composite Minimization with Linear Constraints

9.3.2 Asynchronous Parallel Algorithm for Smooth Minimization

Although the algorithm described in the previous section solves a simple subproblem at each iteration, it is inherently sequential. This can be a disadvantage when addressing large-scale problems. To overcome this concern, we develop an asynchronous parallel method that solves Problem (9.1) for the smooth case.

Our parallel algorithm is similar to Algorithm 21, except for a crucial difference: now we may have multiple processors, and each of these executes the loop 2–6 *independently* without the need for coordination. This way, we can solve subproblems (i.e., multiple pairs) simultaneously in parallel, and due to the asynchronous nature of our algorithm, we can execute updates as they complete, without requiring any locking.

The critical issue, however, with implementing an asynchronous algorithm in the presence of non-separable constraints is ensuring feasibility throughout the course of the algorithm. This requires the operation $x_i \leftarrow x_i + \delta$ to be executed in an *atomic* (i.e., sequentially consistent) fashion. Modern processors facilitate that without an additional locking structure through the "compare-and-swap" instruction [128]. Since the updates use atomic increments and each update satisfies $Ad^k = 0$, the net effect of T updates is $\sum_{k=1}^{T} Ad^k = 0$, which is feasible despite asynchronicity of the algorithm.

The next key issue is that of convergence. In an asynchronous setting, the updates are based on *stale* gradients that are computed using values of x read many iterations earlier. But provided that gradient staleness is bounded, we can establish a sublinear convergence rate of the asynchronous parallel algorithm (Theorem 9.4.2). More formally, we assume that in iteration k, *stale* gradients are computed based on $x^{D(k)}$ such that $k - D(k) \leq \tau$. The bound on staleness, denoted by τ, captures the degree of parallelism in the method: such parameters are typical in asynchronous systems and provides a bound on the delay of the updates [91].

Before concluding the discussion on our asynchronous algorithm, it is important to note the difficulty of extending our algorithm to nonsmooth problems. For example, consider the case where $h = \mathbb{I}_C$ (indicator function of some convex set). Although a pairwise update as suggested above maintains feasibility with respect to the linear constraint $Ax = 0$, it may violate the feasibility of being in the convex set C. This complication can be circumvented by using a convex combination of the current iterate with the update, as this would retain overall feasibility. However, it would complicate the convergence analysis. We plan to investigate this direction in future work.

9.3.3 Stochastic Minimization

An important subclass of Problem (9.1) assumes separable losses $f(x) = \frac{1}{N}\sum_{i=1}^{N} f_i(x)$. This class arises naturally in many machine learning applications where the loss separates over training examples. To take advantage of this added separability of f, we can derive a stochastic block-CD procedure.

Our key innovation here is the following: in addition to randomly picking an edge (i, j), we also pick a function randomly from $\{f_1, \cdots, f_N\}$ and perform our update using this function. This choice substantially reduces the cost of each iteration when N is large, since now the gradient calculations involve only the randomly selected function f_i (i.e., we now use a stochastic-gradient). Pseudocode is given in Algorithm 22.

```
1: Choose $x^0 \in \mathbb{R}^n$ such that $Ax^0 = 0$.
2: for $k \geq 0$ do
3:     Select a random edge $(i_k, j_k) \in E$ with probability $p_{i_k j_k}$
4:     Select random integer $l \in [N]$
5:     $x^{k+1} \leftarrow x^k + U_{i_k j_k} Z(f_l, x^k, (i_k, j_k), \alpha_k / L_{i_k j_k})$
6:     $k \leftarrow k + 1$
7: end for
```

Algorithm 22: Stochastic Minimization with Linear Constraints

Notice that the per iteration cost of Algorithm 22 is lower than Algorithm 21 by a factor of N. However, as we will see later, this speedup comes at a price of slower convergence rate (Theorem 9.4.3). Moreover, to ensure convergence, decaying step sizes $\{\alpha_k\}_{k \geq 0}$ are generally chosen.

9.4 Convergence Analysis

In this section, we outline convergence results for the algorithms described above. The proofs are somewhat technical, and hence left in the appendix; here we present only the key ideas.

For simplicity, we present our analysis for the following reformulation of the main problem:

$$\begin{aligned} \min_{y,z} \quad & f(y,z) + \textstyle\sum_{i=1}^{b} h(y_i, z_i) \\ \text{subject to} \quad & \textstyle\sum_{i=1}^{b} y_i = 0, \end{aligned} \tag{9.4}$$

where $y_i \in \mathbb{R}^{n_y}$ and $z_i \in \mathbb{R}^{n_z}$. Let $y = [y_1^\top \cdots y_b^\top]^\top$ and $z = [z_1^\top \cdots z_b^\top]^\top$. We use x to denote the concatenated vector $[y^\top z^\top]^\top$ and hence we assume (unless otherwise mentioned) that the constraint matrix A is defined as follows

$$A \begin{pmatrix} y \\ z \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^{b} y_i \\ 0 \end{pmatrix}. \tag{9.5}$$

It is worth emphasizing that this analysis *does not* result in any loss of generality. This is due to the fact that Problem (9.1) with a general constraint matrix $\tilde{A}$ having full row-rank submatrices $\tilde{A}_i$'s can be rewritten in the form of Problem (9.4) by using the transformation specified in Section 9.11 of the appendix. It is important to note that this reduction is presented only for the ease of exposition. For our experiments, we directly solve the problem in Equation (9.2).

Let $\eta_k = \{(i_0, j_0), \ldots, (i_{k-1}, j_{k-1})\}$ denote the pairs selected up to iteration $k-1$. To simplify notation, assume (without loss of generality) that the Lipschitz constant for the partial gradient $\nabla_i f(x)$ and $\nabla_{ij} f(x)$ is L for all $i \in [n]$ and $(i,j) \in E$.

Similar to [106], we introduce a Laplacian matrix $\mathcal{L} \in \mathbb{R}^{b \times b}$ that represents the communication graph G. Since we also have unconstrained variables z_i, we introduce a diagonal matrix $\mathcal{D} \in \mathbb{R}^{b \times b}$.

$$\mathcal{L}_{ij} = \begin{cases} \sum_{r \neq i} \frac{p_{ir}}{2L} & i = j \\ -\frac{p_{ij}}{2L} & i \neq j \end{cases} \qquad \mathcal{D}_{ij} = \begin{cases} \frac{p_i}{L} & i = j \\ 0 & i \neq j \end{cases}$$

We use $\mathcal{K}$ to denote the concatenation of the Laplacian $\mathcal{L}$ and the diagonal matrix $\mathcal{D}$. More formally,

$$\mathcal{K} = \begin{bmatrix} \mathcal{L} \otimes I_{n_y} & 0 \\ 0 & \mathcal{D} \otimes I_{n_z} \end{bmatrix}.$$

This matrix induces a norm $\|x\|_{\mathcal{K}} = \sqrt{x^\top \mathcal{K} x}$ on the *feasible subspace*, with a corresponding dual norm

$$\|x\|_{\mathcal{K}}^* = \sqrt{x^\top \left(\begin{bmatrix} \mathcal{L}^+ \otimes I_{n_y} & 0 \\ 0 & \mathcal{D}^{-1} \otimes I_{n_z} \end{bmatrix} \right) x}$$

Let X^* denote the set of optimal solutions and let x^0 denote the initial point. We define the following distance, which quantifies how far the initial point is from the optimal, taking into account the graph layout and edge selection probabilities

$$R(x^0) := \max_{x : f(x) \leq f(x^0)} \max_{x^* \in X^*} \|x - x^*\|_{\mathcal{K}}^* \tag{9.6}$$

Note. Before delving into the details of the convergence results, we would like to draw the reader's attention to the impact of the communication network G on convergence. In general, the convergence results depend on $R(x^0)$, which in turn depends on the Laplacian $\mathcal{L}$ of the graph G. As a rule of thumb, the larger the connectivity of the graph, the smaller the value of $R(x^0)$, and hence, faster the convergence.

9.4.1 Convergence Results for the Smooth Case

We first consider the case when $h = 0$. Here the subproblem at k^{th} iteration has a very simple update $d_{i_k j_k} = U_{i_k} d^k - U_{j_k} d^k$ where $d^k = \frac{\alpha_k}{2L}(\nabla_{j_k} f(x^k) - \nabla_{i_k} f(x^k))$. We now prove that Algorithm 21 attains an $O(1/k)$ convergence rate.

Theorem 9.4.1. *Let $\alpha_k = 1$ for $k \geq 0$, and let $\{x^k\}_{k\geq 0}$ be the sequence generated by Algorithm 21; let f^* denote the optimal value. Then, we have the following rate of convergence:*

$$\mathbb{E}[f(x^k)] - f^* \leq \frac{2R^2(x^0)}{k}$$

where $R(x^0)$ is as defined in Equation (9.6).

Proof Sketch. We first prove that each iteration leads to descent in expectation. More formally, we get

$$\mathbb{E}_{i_k j_k}[f(x^{k+1})|\eta_k] \leq f(x^k) - \tfrac{1}{2}\nabla f(x^k)^\top \mathcal{K} \nabla f(x^k).$$

The above step can be proved using Lemma 9.2.1. Let $\Delta_k = \mathbb{E}[f(x^k)] - f^*$. It can be proved that

$$\frac{1}{\Delta_k} \leq \frac{1}{\Delta_{k+1}} - \frac{1}{2R^2(x^0)}$$

This follows from the fact that

$$\begin{aligned} f(x^{k+1}) - f^* &\leq \|x^k - x^*\|_{\mathcal{K}}^* \|\nabla f(x^k)\|_{\mathcal{K}} \\ &\leq R(x^0)\|\nabla f(x^k)\|_{\mathcal{K}} \quad \forall k \geq 0 \end{aligned}$$

Telescoping the sum, we get the desired result. □

Note that Theorem 9.4.1 is a strict generalization of the analysis in [106] and [105] due to: (i) the presence of unconstrained variables z; and (ii) the presence of a non-decomposable objective function. it is also worth emphasizing that our convergence rates improve upon those of [105], since they do not involve an exponential dependence of the form b^m on the number of constraints.

We now turn our attention towards the convergence analysis of our asynchronous algorithm under a consistent reading model [91]. In this context we would like to emphasize that while our theoretical analysis assumes consistent reads, we do not enforce this assumption in our experiments.

Theorem 9.4.2. *Let $\rho > 1$ and $\alpha_k = \alpha$ be such that $\alpha < 2/(1+\tau+\tau\rho^\tau)$ and $\alpha < (\rho - 1)/(\sqrt{2}(\tau+2)(\rho^{\tau+1}+\rho))$. Let $\{x_k\}_{k\geq 0}$ be the sequence generated by asynchronous algorithm using step size α_k and let f^* denote the optimal value. Then, we have the following rate of convergence for the expected values of the objective function*

$$\mathbb{E}[f(x_k)] - f^* \leq \frac{R^2(x^0)}{\mu k}$$

where $R(x^0)$ is as defined in Equation (9.6) *and $\mu = \frac{\alpha_k^2}{2}\left(\frac{1}{\alpha_k} - \frac{1+\tau+\tau\rho^\tau}{2}\right)$.*

Proof Sketch. For ease of exposition, we describe the case where the unconstrained variables z are absent. The analysis of case with z variables can be carried out in a similar manner. Let $D(k)$ denote the iterate of the variables used in the k^{th} iteration (the existence of $D(k)$ follows from the consistent reading assumption). Let

$$d^k = \frac{\alpha_k}{2L}\left(\nabla_{y_{j_k}} f(x^{D(k)}) - \nabla_{y_{i_k}} f(x^{D(k)})\right)$$

and $d^k_{i_k j_k} = x^{k+1} - x^k = U_{i_k} d^k - U_{j_k} d^k$. Using Lemma 9.2.1 and the assumption that staleness in the variables is bounded by τ, i.e., $k - D(k) \leq \tau$ and definition of d^k_{ij}, we can derive the following bound:

$$\mathbb{E}[f(x^{k+1})] \leq \mathbb{E}[f(x^k)] - L\left(\frac{1}{\alpha_k} - \frac{1+\tau}{2}\right)\mathbb{E}[\|d^k_{i_k j_k}\|^2] + \frac{L}{2}\mathbb{E}\left[\sum_{t=1}^{\tau} \|d^{k-t}_{i_{k-t} j_{k-t}}\|^2\right].$$

In order to obtain an upper bound on the norms of $d^k_{i_k j_k}$, we prove that

$$\mathbb{E}\left[\|d^{k-1}_{i_{k-1} j_{k-1}}\|^2\right] \leq \rho \mathbb{E}\left[\|d^k_{i_k j_k}\|^2\right]$$

This can proven using mathematical induction. Using the above bound on $\|d^k_{i_k j_k}\|^2$, we get

$$\mathbb{E}[f(x^{k+1})] \leq \mathbb{E}[f(x^k)] - L\left(\frac{1}{\alpha_k} - \frac{1+\tau+\tau\rho^\tau}{2}\right)\mathbb{E}[\|d^k_{i_k j_k}\|^2]$$

This proves that the method is a descent method in expectation. Following similar analysis as Theorem 9.4.1, we get the required result. □

Note the dependence of convergence rate on the staleness bound τ. For larger values of τ, the stepsize α_k needs to be decreased to ensure convergence, which in turn slows down the convergence rate of the algorithm. Nevertheless, the convergence rate remains $O(1/k)$.

The last smooth case we analyze is our stochastic algorithm.

Theorem 9.4.3. *Let $\alpha_i = \sqrt{\Delta_0 L}/(M\sqrt{i+1})$ for $i \geq 0$ in Algorithm 22. Let $\{x^k\}_{k\geq 0}$ be the sequence generated by Algorithm 22 and let f^* denote the optimal value. We denote $\bar{x}^k = \arg\min_{0\leq i\leq k} f(x^k)$. Then, we have the following rate of convergence for the expected values of the objective:*

$$\mathbb{E}[f(\bar{x}^k)] - f^* \leq O\left(\frac{1}{\sqrt[4]{k}}\right)$$

where $\Delta_0 = f(x^0) - f^*$.

The convergence rate is $O(1/k^{1/4})$ as opposed to $O(1/k)$ of Theorem 9.4.1. On the other hand, the iteration complexity is lower by a factor of N; this kind of tradeoff is typical in stochastic algorithms, where the slower rate is the price we pay for a lower iteration complexity. We believe that the convergence rate can be improved to $O(1/\sqrt{k})$, the rate generally observed in stochastic algorithms, by a more careful analysis.

9.4.2 Nonsmooth Case

We finally state the convergence rate for the nonsmooth case ($h \not\equiv 0$) in the case of a sum constraint. Similar to [105], we assume h is coordinatewise separable (i.e. we can write $h(x) = \sum_{i=1}^{b} \sum_j x_{ij}$), where x_{ij} is the j^{th} coordinate in the i^{th} block. For this analysis, we assume that the graph G is a clique [1] with uniform probability, i.e., $\lambda = p_{ij} = 2/b(b-1)$.

Theorem 9.4.4. *Assume $Ax = \sum_i A_i x_i$. Let $\{x^k\}_{k\geq 0}$ be the sequence generated by Algorithm 21 and let F^* denote the optimal value. Assume that the graph G is a clique with uniform probability. Then we have the following:*

$$\mathbb{E}[F(x^k) - F^*] \leq \frac{b^2 L R^2(x^0)}{2k + \frac{b^2 L R^2(x^0)}{\Delta_0}},$$

where $R(x^0)$ is as defined in Equation (9.6)*.*

This convergence rate is a generalization of the convergence rate obtained in Necoara and Patrascu [105] for a single linear constraint (see Theorem 1 in [105]). It is also an improvement of the rate obtained in Necoara and Patrascu [105] for general linear constraints (see Theorem 4 in [105]) when applied to the special case of a sum constraint. Our improvement comes in the form of a tractable constant, as opposed to the exponential dependence $O(b^m)$ shown in [105].

9.5 Applications

To gain a better understanding of our approach, we state some applications of interest, while discussing details of Algorithm 21 and Algorithm 22 for them. While there are

[1] We believe our results also easily extend to the general case along the lines of [145, 146, 147], using the concept of *Expected Separable Overapproximation* (ESO). Moreover, the assumption is not totally impractical, e.g., in a multicore setting with a zero-sum constraint (i.e. $A_i = I$), the clique-assumption introduces little cost.

many applications of problem (9.1), we only mention a few prominent ones here.
Support Vector Machines: The SVM dual (with bias term) assumes the form (9.1); specifically,

$$\begin{aligned} &\min_{\alpha} \tfrac{1}{2} \textstyle\sum_{i,j} \alpha_i \alpha_j y_i y_j z_i^\top z_j - \sum_{i=1}^{n} \alpha_i \\ &s.t. \textstyle\sum_i \alpha_i y_i = 0, \;\; 0 \leq \alpha_i \leq C \quad \forall\, i \in [n]. \end{aligned} \tag{9.7}$$

Here, z_i denotes the feature vector of the i^{th} training example and $y_i \in \{1, -1\}$ denotes the corresponding label. By letting $f(\alpha) = \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j z_i^\top z_j - \sum_i \alpha_i$ and $h(\alpha) = \sum_i \mathbb{I}(0 \leq \alpha_i \leq C)$ and $A = [y_1, \ldots, y_n]$ this problem can be written in form of Problem (9.1). Using Algorithm 21 for SVM involves solving a sub-problem similar to one used in SMO in the scalar case (i.e., $\alpha_i \in \mathbb{R}$) and can be solved in linear time in the block case (see [14]).
Generalized Lasso: The objective is to solve the following optimization problem.

$$\min_\beta \quad \tfrac{1}{2}\|Y - X\beta\|_2^2 + \lambda \|D\beta\|_1$$

where $Y \in \mathbb{R}^N$ denotes the output, $X \in \mathbb{R}^{N \times n}$ is the input and $D \in \mathbb{R}^{q \times n}$ represents a specified penalty matrix. This problem can also be seen as a specific case of Problem (9.1) by introducing an auxiliary variable t and slack variables u, v. Then, $f(\beta, t) = \frac{1}{2}\|Y - X\beta\|_2^2 + \sum_i t_i$, $h(u, v) = \mathbb{I}(u \geq 0) + \mathbb{I}(v \geq 0)$ and, $t - D\beta - u = 0$ and $t + D\beta - v = 0$ are the linear constraints. To solve this problem, we can use either Algorithm 21 or Algorithm 22. In general, optimization of convex functions on a structured convex polytope can be solved in a similar manner.
Unconstrained Separable Optimization: As mentioned earlier, another interesting application is for unconstrained separable optimization. For any problem $\min_x \sum_i f_i(x)$—a form generally encountered across machine learning—can be rewritten using variable-splitting. Solving the problem in distributed environment requires considerable synchronization (for the consensus constraint), which can slow down the algorithm significantly. However, the dual of the problem is

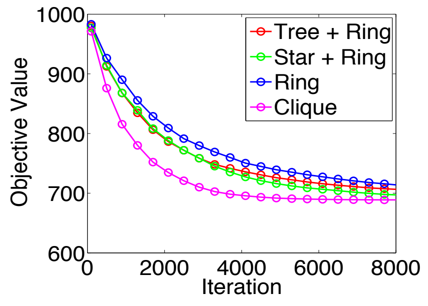
$$\min_{\lambda} \sum_i f_i^*(\lambda_i) \quad s.t \;\; \textstyle\sum_{i=1}^{N} \lambda_i = 0.$$

where f_i^* is the Fenchel conjugate of f_i. This reformulation perfectly fits our framework and can be solved in an asynchronous manner using the procedure described in Section 9.3.2.

Other interesting application include constrained least square problem, multi-agent planning problems, resource allocation—see [105, 106] and references therein for more examples.

9.6 Experiments

In this section, we present our empirical results. In particular, we examine the behavior of random coordinate descent algorithms analyzed in this chapter under different



h

Figure 9.1: Objective value vs. number of iterations for different graph topologies. Note that larger the connectivity of the graph, faster is the convergence.

communication constraints and concurrency conditions. [2]

9.6.1 Effect of Communication Constraints

Our first set of experiments test the affect of the connectivity of the graph on the convergence rate. In particular, recall that the convergence analysis established in Theorem 9.4.1 depends on the Laplacian of the communication graph. In this experiment we demonstrate how communication constraints affect convergence in practice. We experiment with the following graph topologies of graph G: Ring, Clique, Star + Ring (i.e., the union of edges of a star and a ring) and Tree + Ring. On each layout we run the sequential Algorithm 21 on the following quadratic problem

$$\begin{aligned} \min \quad & C \textstyle\sum_{i=1}^{N} \|x_i - (i \bmod 10)\mathbf{1}\|^2 \\ s.t. \quad & \textstyle\sum_{i=1}^{N} A_i x_i = 0, \end{aligned} \tag{9.8}$$

Note the decomposable structure of the problem. For this experiment, we use $N = 1000$ and $x_i \in \mathbb{R}^{50}$. We have 10 constraints whose coefficients are randomly generated from $U[0,1]$ and we choose C such that the objective evaluates to 1000 when $x = 0$.

The results for Algorithm 21 on each topology for 10000 iterations are shown in Figure 9.1. The results clearly show that better connectivity implies better convergence rate. Note that while the clique topology has significantly better convergence than other topologies, acceptable long-term performance can be achieved by much sparser topologies such as Star + Ring and Tree + Ring.

Having a sparse communication graph is important to lower the cost of a distributed system. Furthermore, it is worth mentioning that the sparsity of the communication graph is also important in a multicore setting; since Algorithm 21 requires computing $(A_i A_i^\top + A_j A_j^\top)^+$ for each communicating pair of nodes (i, j). Our analysis shows that this computation takes a significant portion of the running time and hence it is essential to minimize the number of variable pairs that are allowed to be updated.

[2]All experiments were conducted on a Google Compute Engine virtual machine of type "n1-highcpu-16", which comprises 16 virtual CPUs and 14.4 GB of memory. For more details, please refer to https://cloud.google.com/compute/docs/machine-types#highcpu.

9.6.2 Concurrency and Synchronization

As seen earlier, compared to Tree + Ring, Star + Ring is a low diameter layout (diameter = 2). Hence, in a sequential setting, it indeed results in a faster convergence. However, Star + Ring requires a node to be connected to all other nodes. This high-degree node could be a contention point in a parallel setting. We test the performance of our asynchronous algorithm in this setting. To assess how the performance would be affected with such contention and how asynchronous updates would increase performance, we conduct another experiment on the synthetic problem (9.8) but on a larger scale ($N = 10000$, $x_i \in \mathbb{R}^{100}$, 100 constraints).

Our concurrent update follows a master/slave scheme. Each thread performs a loop where in each iteration it elects a master i and slave j and then applies the following sequence of actions:

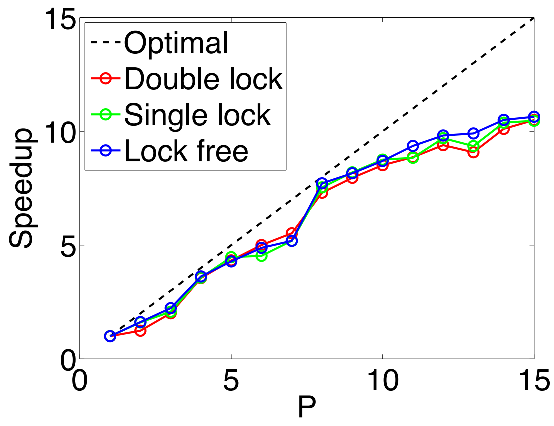
1. Obtain the information required for the update from the master (i.e., information for calculating the gradients used for solving the subproblem).
2. Send the master information to the slave, update the slave variable and get back the information needed to update the master.
3. Update the master based (only) on the information obtained from steps 1 and 2.

We emphasize that the master is not allowed to read its own state at step 3 except to apply an increment, which is computed based on steps 1 and 2. This ensures that the master's increment is consistent with that of the slave, even if one or both of them was being concurrently overwritten by another thread. More details on the implementation can be found in [58].

Given this update scheme, we experiment with three levels of synchronization: (a) **Double Locking:** Locks the master and the slave through the entire update. Because the objective function is decomposable, a more conservative locking (e.g. locking all nodes) is not needed. (b) **Single Locking:** Locks the master during steps 1 and 3 (the master is unlocked during step 2 and locks the slave during step 2). (c) **Lock-free:** No locks are used. Master and slave variables are updated through atomic increments similar to Hogwild! method.

Following [128], we use spinlocks instead of mutex locks to implement locking. Spinlocks are preferred over mutex locks when the resource is locked for a short period of time, which is the case in our algorithm. For each locking mechanism, we vary the number of threads from 1 to 15. We stop when $f_0 - f_t > 0.99(f_0 - f^*)$, where f^* is computed beforehand up to three significant digits. Similar to [128], we add artificial delay to steps 1 and 2 in the update scheme to model complicated gradient calculations and/or network latency in a distributed setting.

Figure 9.2 shows the speedup for Tree + Ring and Star + Ring layouts. The figure clearly shows that a fully synchronous method suffers from contention in the Star + Ring topology whereas asynchronous method does not suffer from this problem and hence, achieves higher speedups. Although the Tree + Ring layouts achieves higher speedup than Star + Ring, the latter topology results in much less running time ($\sim$ 67 seconds vs 91 seconds using 15 threads).



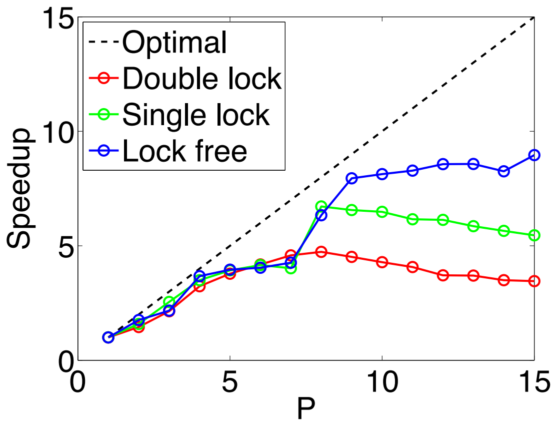


Figure 9.2: Speedup for Tree + Ring (top) and Star + Ring (bottom) topologies and different levels of synchronization. Note for Star + Ring topology, speedup of asynchronous algorithm is significantly higher than that of synchronous version.

9.6.3 Practical Case Study: Parallel Training of Linear SVM

In this section, we explore the effect of parallelism on randomized CD for training a linear SVM based on the dual formulation stated in (9.7). Necoara et. al. [105] have shown that, in terms of CPU time, a sequential randomized CD outperforms coordinate descent using Gauss-Southwell selection rule. It was also observed that randomized CD outperforms LIBSVM [24] for large datasets while maintaining reasonable performance for small datasets.

In this experiment we use a clique layout. For SVM training in a multicore setting, using a clique layout does not introduce additional cost compared to a more sparse layout. To maintain the box constraint, we use the double-locking scheme described in Section 9.6.2 for updating a pair of dual variables.

One advantage of coordinate descent algorithms is that they do not require the storage of the Gram matrix; instead they can compute its elements on the fly. That comes, however, at the expense of CPU time. Similar to [105], to speed up gradient computations without increasing memory requirements, we maintain the primal weight vector of the linear SVM and use it to compute gradients. Basically, if we increment α_i by δ_i and α_j by δ_j, then we increment the weight vector by $\delta_i y_i x_i + \delta_j y_j x_j$. This increment is accomplished using atomic additions. However, this implies that all threads will be concurrently updating the primal weight vector. Similar to [128], we require these updates to be sparse with small overlap between non-zero coordinates in order to ensure convergence. In other words, we require training examples to have sparse features with small overlap between non-zero features.

We report speedups on two datasets used in [105].[3] For each dataset, we train the SVM model until $f_0 - f_t > 0.9999(f_0 - f^*)$, where f^* is the objective reported in [105].

[3]Datasets can be downloaded from http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets.

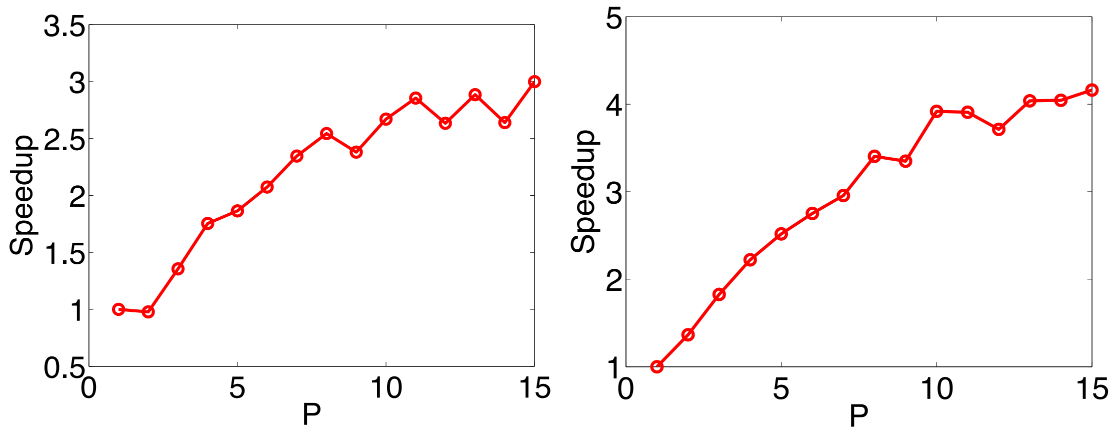


Figure 9.3: Speedup for linear SVM training on `a7a` (top) and `w8a` datasets.

In Figure 9.3, we report speedup for both the datasets. The figure shows that parallelism indeed increases the performance of randomized CD training of linear SVM.

Appendix: Omitted Proofs

9.7 Proof of Theorem 9.4.1

Proof. Taking the expectation over the choice of edges (i_k, j_k) gives the following inequality

$$\begin{aligned}
&\mathbb{E}_{i_k j_k}[f(x^{k+1})|\eta_k] \\
&\leq \mathbb{E}_{i_k j_k}\left[f(x^k) - \frac{1}{4L}\|\nabla_{y_{i_k}} f(x^k) - \nabla_{y_{j_k}} f(x^k)\|^2 - \frac{1}{2L}\|\nabla_{z_{i_k}} f(x^k)\|^2 - \frac{1}{2L}\|\nabla_{z_{j_k}} f(x^k)\|^2\right] \\
&\leq f(x^k) - \frac{1}{2}\nabla_y f(x^k)^\top (\mathcal{L} \otimes I_{n_y})\nabla_y f(x^k) - \frac{1}{2}\nabla_z f(x^k)^\top (\mathcal{D} \otimes I_{n_z})\nabla_z f(x^k) \\
&\leq f(x^k) - \frac{1}{2}\nabla f(x^k)^\top \mathcal{K} \nabla f(x^k), \qquad (9.9)
\end{aligned}$$

where $\otimes$ denotes the Kronecker product. This shows that the method is a descent method. Now we are ready to prove the main convergence theorem. We have the following:

$$\begin{aligned}
f(x^{k+1}) - f^* &\leq \langle \nabla f(x^k), x^k - x^* \rangle \leq \|x^k - x^*\|_{\mathcal{K}}^* \|\nabla f(x^k)\|_{\mathcal{K}} \\
&\leq R(x^0)\|\nabla f(x^k)\|_{\mathcal{K}} \quad \forall k \geq 0.
\end{aligned}$$

Combining this with inequality (9.9), we obtain

$$\mathbb{E}[f(x^{k+1}|\eta_k] \leq f(x^k) - \frac{(f(x^k) - f^*)^2}{2R^2(x^0)}.$$

Taking the expectation of both sides an denoting $\Delta_k = \mathbb{E}[f(x^k)] - f^*$ gives

$$\Delta_{k+1} \le \Delta_k - \frac{\Delta_k^2}{2R^2(x^0)}.$$

Dividing both sides by $\Delta_k \Delta_{k+1}$ and using the fact that $\Delta_{k+1} \le \Delta_k$ we obtain

$$\frac{1}{\Delta_k} \le \frac{1}{\Delta_{k+1}} - \frac{1}{2R^2(x^0)}.$$

Adding these inequalities for k steps $0 \le \frac{1}{\Delta_0} \le \frac{1}{\Delta_k} - \frac{k}{2R^2(x^0)}$ from which we obtain the statement of the theorem where $C = 2R^2(x^0)$. $\square$

9.8 Proof of Theorem 9.4.3

Proof. In this case, the expectation should be over the selection of the pair (i_k, j_k) and random index $l_k \in [N]$. In this proof, the definition of η_k includes l_k i.e., we have $\eta_k = \{(i_0, j_0, l_0), \dots, (i_{k-1}, j_{k-1}, l_{k-1})\}$. We define the following:

$$\begin{aligned}
d_{i_k}^k &= \left[\frac{\alpha_k}{2L} \left[\nabla_{y_{j_k}} f_{l_k}(x^k) - \nabla_{y_{i_k}} f_{l_k}(x^k) \right]^\top, \quad -\frac{\alpha_k}{L} \left[\nabla_{z_{i_k}} f_{l_k}(x^k) \right]^\top \right]^\top, \\
d_{j_k}^k &= \left[\frac{\alpha_k}{2L} \left[\nabla_{y_{j_k}} f_{l_k}(x^k) - \nabla_{y_{i_k}} f_{l_k}(x^k) \right]^\top, \quad \frac{\alpha_k}{L} \left[\nabla_{z_{j_k}} f_{l_k}(x^k) \right]^\top \right]^\top, \\
d_{i_k j_k}^{l_k} &= U_{i_k} d_{i_k}^k - U_{j_k} d_{j_k}^k.
\end{aligned}$$

For the expectation of objective value at x^{k+1}, we have

$$\begin{aligned}
&\mathbb{E}[f(x^{k+1}) | \eta_k] \le \mathbb{E}_{i_k j_k} \mathbb{E}_{l_k} \left[f(x^k) + \left\langle \nabla f(x^k), d_{i_k j_k}^{l_k} \right\rangle + \frac{L}{2} \| d_{i_k j_k}^{l_k} \|^2 \right] \\
&\le \mathbb{E}_{i_k j_k} \left[f(x^k) + \left\langle \nabla f(x^k), \mathbb{E}_{l_k}[d_{i_k j_k}^{l_k}] \right\rangle + \frac{L}{2} \mathbb{E}_{l_k}[\| d_{i_k j_k}^{l_k} \|^2] \right] \\
&\le \mathbb{E}_{i_k j_k} \Big[f(x^k) + \frac{\alpha_k}{2L} \left\langle \nabla_{y_{i_k}} f(x^k), \mathbb{E}_{l_k}[\nabla_{y_{j_k}} f_{l_k}(x^k) - \nabla_{y_{i_k}} f_{l_k}(x^k)] \right\rangle \\
&\qquad + \frac{\alpha_k}{2L} \left\langle \nabla_{y_{j_k}} f(x^k), \mathbb{E}_{l_k}[\nabla_{y_{i_k}} f_{l_k}(x^k) - \nabla_{y_{j_k}} f_{l_k}(x^k)] \right\rangle \\
&\qquad - \frac{\alpha_k}{L} \left\langle \nabla_{z_{i_k}} f(x^k), \mathbb{E}_{l_k}[\nabla_{z_{i_k}} f_{l_k}(x^k)] \right\rangle - \frac{\alpha_k}{L} \left\langle \nabla_{z_{j_k}} f(x^k), \mathbb{E}_{l_k}[\nabla_{z_{j_k}} f_{l_k}(x^k)] \right\rangle + \frac{L}{2} \mathbb{E}_{l_k}[\| d_{i_k j_k}^{l_k} \|^2] \Big].
\end{aligned}$$

Taking expectation over l_k, we get the following relationship:

$$\begin{aligned}
\mathbb{E}[f(x^{k+1}) | \eta_k] \le \mathbb{E}_{i_k j_k} \Big[& f(x^k) + \frac{\alpha_k}{2L} \left\langle \nabla_{y_{i_k}} f(x^k), \nabla_{y_{j_k}} f(x^k) - \nabla_{y_{i_k}} f(x^k) \right\rangle \\
& + \frac{\alpha_k}{2L} \left\langle \nabla_{y_{j_k}} f(x^k), \nabla_{y_{i_k}} f(x^k) - \nabla_{y_{j_k}} f(x^k) \right\rangle \\
& - \frac{\alpha_k}{L} \left\langle \nabla_{z_{i_k}} f(x^k), \nabla_{z_{i_k}} f(x^k) \right\rangle - \frac{\alpha_k}{L} \left\langle \nabla_{z_{j_k}} f(x^k), \nabla_{z_{j_k}} f(x^k) \right\rangle + \frac{L}{2} \mathbb{E}_{l_k}[\| d_{i_k j_k}^{l_k} \|^2] \Big].
\end{aligned}$$

We first note that $\mathbb{E}_{l_k}[\|d^{l_k}_{i_kj_k}\|^2] \leq 8M^2\alpha_k^2/L^2$ since $\|\nabla f_l\| \leq M$. Substituting this in the above inequality and simplifying we get,

$$\begin{aligned}\mathbb{E}[f(x_{k+1})|\eta_k] &\leq f(x^k) - \alpha_k \nabla_y f(x^k)^\top (\mathcal{L} \otimes I_n) \nabla_y f(x^k) - \alpha_k \nabla_z f(x^k)^\top (\mathcal{D} \otimes I_n) \nabla_z f(x^k) + \frac{4M^2\alpha_k^2}{L} \\ &\leq f(x^k) - \alpha_k \nabla f(x^k)^\top \mathcal{K} \nabla f(x^k) + \frac{4M^2\alpha_k^2}{L}. \qquad (9.10)\end{aligned}$$

Similar to Theorem 9.4.1, we obtain a lower bound on $\nabla f(x^k)^\top \mathcal{K} \nabla f(x^k)$ in the following manner.

$$\begin{aligned}f(x^k) - f^* \leq \langle \nabla f(x^k), x^k - x^* \rangle &\leq \|x^k - x^*\|^*_{\mathcal{K}}.\|\nabla f(x^k)\|_{\mathcal{K}} \\ &\leq R(x^0)\|\nabla f(x^k)\|_{\mathcal{K}}.\end{aligned}$$

Combining this with inequality Equation (9.10), we obtain

$$\mathbb{E}[f(x_{k+1})|\eta_k] \leq f(x^k) - \alpha_k \frac{(f(x^k) - f^*)^2}{R^2(x^0)} + \frac{4M^2\alpha_k^2}{L}.$$

Taking the expectation of both sides an denoting $\Delta_k = \mathbb{E}[f(x^k)] - f^*$ gives

$$\Delta_{k+1} \leq \Delta_k - \alpha_k \frac{\Delta_k^2}{R^2(x^0)} + \frac{4M^2\alpha_k^2}{L}.$$

Adding these inequalities from $i = 0$ to $i = k$ and use telescopy we get,

$$\Delta_{k+1} + \sum_{i=0}^{k} \alpha_i \frac{\Delta_k^2}{R^2(x^0)} \leq \Delta_0 + \frac{4M^2}{L} \sum_{i=0}^{k} \alpha_i^2.$$

Using the definition of $\bar{x}_{k+1} = \arg\min_{0 \leq i \leq k+1} f(x_i)$, we get

$$\sum_{i=0}^{k} \alpha_i \frac{(\mathbb{E}[f(\bar{x}_{k+1}) - f^*])^2}{R^2(x^0)} \leq \Delta_{k+1} + \sum_{i=0}^{k} \alpha_i \frac{\Delta_k^2}{R^2(x^0)} \leq \Delta_0 + \frac{4M^2}{L} \sum_{i=0}^{k} \alpha_i^2.$$

Therefore, from the above inequality we have,

$$\mathbb{E}[f(\bar{x}_{k+1}) - f^*] \leq R(x^0) \sqrt{\frac{(\Delta_0 + 4M^2 \sum_{i=0}^{k} \alpha_i^2 / L)}{\sum_{i=0}^{k} \alpha_i}}.$$

Note that $\mathbb{E}[f(\bar{x}_{k+1}) - f^*] \to 0$ if we choose step sizes satisfying the condition that $\sum_{i=0}^{\infty} \alpha_i = \infty$ and $\sum_{i=0}^{\infty} \alpha_i^2 < \infty$. Substituting $\alpha_i = \sqrt{\Delta_0 L}/(2M\sqrt{i+1})$, we get the required result using the reasoning from [109] (we refer the reader to Section 2.2 of [109] for more details). □

9.9 Proof of Theorem 9.4.2

Proof. For ease of exposition, we analyze the case where the unconstrained variables z are absent. The analysis of case with z variables can be carried out in a similar manner. Consider the update on edge (i_k, j_k). Recall that $D(k)$ denotes the index of the iterate used in the k^{th} iteration for calculating the gradients. Let $d^k = \frac{\alpha_k}{2L}\left(\nabla_{y_{j_k}} f(x^{D(k)}) - \nabla_{y_{i_k}} f(x^{D(k)})\right)$ and $d^k_{i_k j_k} = x^{k+1} - x^k = U_{i_k} d^k - U_{j_k} d^k$. Note that $\|d^k_{i_k j_k}\|^2 = 2\|d^k\|^2$. Since f is Lipschitz continuous gradient, we have

$$\begin{aligned}
f(x^{k+1}) &\leq f(x^k) + \left\langle \nabla_{y_{i_k} y_{j_k}} f(x^k), d^k_{i_k j_k} \right\rangle + \frac{L}{2}\|d^k_{i_k j_k}\|^2 \\
&\leq f(x^k) + \left\langle \nabla_{y_{i_k} y_{j_k}} f(x^{D(k)}) + \nabla_{y_{i_k} y_{j_k}} f(x^k) - \nabla_{y_{i_k} y_{j_k}} f(x^{D(k)}), d^k_{i_k j_k} \right\rangle + \frac{L}{2}\|d^k_{i_k j_k}\|^2 \\
&\leq f(x^k) - \frac{L}{\alpha_k}\|d^k_{i_k j_k}\|^2 + \left\langle \nabla_{y_{i_k} y_{j_k}} f(x^k) - \nabla_{y_{i_k} y_{j_k}} f(x^{D(k)}), d^k_{i_k j_k} \right\rangle + \frac{L}{2}\|d^k_{i_k j_k}\|^2 \\
&\leq f(x^k) - L\left(\frac{1}{\alpha_k} - \frac{1}{2}\right)\|d^k_{i_k j_k}\|^2 + \|\nabla_{y_{i_k} y_{j_k}} f(x^k) - \nabla_{y_{i_k} y_{j_k}} f(x^{D(k)})\|\|d^k_{i_k j_k}\| \\
&\leq f(x^k) - L\left(\frac{1}{\alpha_k} - \frac{1}{2}\right)\|d^k_{i_k j_k}\|^2 + L\|x^k - x^{D(k)}\|\|d^k_{i_k j_k}\|.
\end{aligned}$$

The third and fourth steps in the above derivation follow from definition of d^k_{ij} and Cauchy-Schwarz inequality respectively. The last step follows from the fact the gradients are Lipschitz continuous. Using the assumption that staleness in the variables is bounded by τ, i.e., $k - D(k) \leq \tau$ and definition of d^k_{ij}, we have

$$\begin{aligned}
f(x^{k+1}) &\leq f(x^k) - L\left(\frac{1}{\alpha_k} - \frac{1}{2}\right)\|d^k_{i_k j_k}\|^2 + L\left(\sum_{t=1}^{\tau} \|d^{k-t}_{i_{k-t} j_{k-t}}\|\|d^k_{i_k j_k}\|\right) \\
&\leq f(x^k) - L\left(\frac{1}{\alpha_k} - \frac{1}{2}\right)\|d^k_{i_k j_k}\|^2 + \frac{L}{2}\left(\sum_{t=1}^{\tau}\left[\|d^{k-t}_{i_{k-t} j_{k-t}}\|^2 + \|d^k_{i_k j_k}\|^2\right]\right) \\
&\leq f(x^k) - L\left(\frac{1}{\alpha_k} - \frac{1+\tau}{2}\right)\|d^k_{i_k j_k}\|^2 + \frac{L}{2}\sum_{t=1}^{\tau}\|d^{k-t}_{i_{k-t} j_{k-t}}\|^2.
\end{aligned}$$

The first step follows from triangle inequality. The second inequality follows from fact that $ab \leq (a^2 + b^2)/2$. Using expectation over the edges, we have

$$\mathbb{E}[f(x^{k+1})] \leq \mathbb{E}[f(x^k)] - L\left(\frac{1}{\alpha_k} - \frac{1+\tau}{2}\right)\mathbb{E}[\|d^k_{i_k j_k}\|^2] + \frac{L}{2}\mathbb{E}\left[\sum_{t=1}^{\tau}\|d^{k-t}_{i_{k-t} j_{k-t}}\|^2\right]. \quad (9.11)$$

We now prove that, for all $k \geq 0$

$$\mathbb{E}\left[\|d^{k-1}_{i_{k-1} j_{k-1}}\|^2\right] \leq \rho\mathbb{E}\left[\|d^k_{i_k j_k}\|^2\right], \quad (9.12)$$

where we define $\mathbb{E}\left[\|d^{k-1}_{i_{k-1}j_{k-1}}\|^2\right] = 0$ for $k = 0$. Let w^t denote the vector of size $|E|$ such that $w^t_{ij} = \sqrt{p_{ij}}\|d^t_{ij}\|$ (with slight abuse of notation, we use w^t_{ij} to denote the entry corresponding to edge (i, j)). Note that $\mathbb{E}\left[\|d^t_{i_tj_t}\|^2\right] = \mathbb{E}[\|w^t\|^2]$. We prove Equation (9.12) by induction.

Let u^k be a vector of size $|E|$ such that $u^k_{ij} = \sqrt{p_{ij}}\|d^k_{ij} - d^{k-1}_{ij}\|$. Consider the following:

$$\begin{aligned}
\mathbb{E}[\|w^{k-1}\|]^2 - \mathbb{E}[\|w^k\|^2] &= \mathbb{E}[2\|w^{k-1}\|]^2 - \mathbb{E}[\|w^k\|^2 + \|w^{k-1}\|^2] \\
&\leq 2\mathbb{E}[\|w^{k-1}\|^2] - 2\mathbb{E}[\langle w^{k-1}, w^k\rangle] \\
&\leq 2\mathbb{E}[\|w^{k-1}\|\|w^{k-1} - w^k\|] \\
&\leq 2\mathbb{E}[\|w^{k-1}\|\|u^k\|] \leq 2\mathbb{E}[\|w^{k-1}\|\sqrt{2}\alpha_k\|x^{D(k)} - x^{D(k-1)}\|] \\
&\leq \sqrt{2}\alpha_k \sum_{t=\min(D(k-1),D(k))}^{\max(D(k-1),D(k))} \left(\mathbb{E}[\|w^{k-1}\|^2] + \mathbb{E}[\|d^t_{i_tj_t}\|^2]\right). \quad (9.13)
\end{aligned}$$

The fourth step follows from the bound below on $|u^k_{ij}|$

$$\begin{aligned}
|u^k_{ij}| &= \sqrt{p_{ij}}\|d^k_{ij} - d^{k-1}_{ij}\| \\
&\leq \sqrt{p_{ij}}\|(U_i - U_j)\frac{\alpha_k}{2L}(\nabla_{y_i} f(x^{D(k)}) - \nabla_{y_j} f(x^{D(k)}) + \nabla_{y_j} f(x^{D(k-1)}) - \nabla_{y_i} f(x^{D(k-1)}))\| \\
&\leq \sqrt{2p_{ij}}\alpha_k\|x^{D(k)} - x^{D(k-1)}\|.
\end{aligned}$$

The fifth step follows from triangle inequality. We now prove (9.12): the induction hypothesis is trivially true for $k = 0$. Assume it is true for some $k - 1 \geq 0$. Now using Equation (9.13), we have

$$\mathbb{E}[\|w^{k-1}\|]^2 - \mathbb{E}[\|w^k\|^2] \leq \sqrt{2}\alpha_k(\tau + 2)\mathbb{E}[\|w^{k-1}\|^2] + \sqrt{2}\alpha_k(\tau + 2)\rho^{\tau+1}\mathbb{E}[\|w^k\|^2]$$

for our choice of α_k. The last step follows from the fact that $\mathbb{E}[\|d^t_{i_tj_t}\|^2] = \mathbb{E}[\|w^t\|^2]$ and mathematical induction. From the above, we get

$$\mathbb{E}[\|w^{k-1}\|^2] \leq \frac{1 + \sqrt{2}\alpha_k(\tau + 2)\rho^{(\tau+1)}}{1 - \sqrt{2}\alpha_k(\tau + 2)}\mathbb{E}[\|w^k\|^2] \leq \rho\mathbb{E}[\|w^k\|^2].$$

Thus, the statement holds for k. Therefore, the statement holds for all $k \in \mathbb{N}$ by mathematical induction. Substituting the above in Equation (9.11), we get

$$\mathbb{E}[f(x^{k+1})] \leq \mathbb{E}[f(x^k)] - L\left(\frac{1}{\alpha_k} - \frac{1 + \tau + \tau\rho^\tau}{2}\right)\mathbb{E}[\|d^k_{i_kj_k}\|^2].$$

This proves that the method is a descent method in expectation. Using the definition of

d_{ij}^k, we have

$$\begin{aligned}
\mathbb{E}[f(x^{k+1})] &\leq \mathbb{E}[f(x^k)] - \frac{\alpha_k^2}{4L}\left(\frac{1}{\alpha_k} - \frac{1+\tau+\tau\rho^\tau}{2}\right)\mathbb{E}[\|\nabla_{y_{i_k}} f(x^{D(k)}) - \nabla_{y_{j_k}} f(x^{D(k)})\|^2] \\
&\leq \mathbb{E}[f(x^k)] - \frac{\alpha_k^2}{4L}\left(\frac{1}{\alpha_k} - \frac{1+\tau+\tau\rho^\tau}{2}\right)\mathbb{E}[\|\nabla f(x^{D(k)}) - \nabla f(x^{D(k)})\|_{\mathcal{K}}^2] \\
&\leq \mathbb{E}[f(x^k)] - \frac{\alpha_k^2}{2R^2(x^0)}\left(\frac{1}{\alpha_k} - \frac{1+\tau+\tau\rho^\tau}{2}\right)\mathbb{E}[(f(x^{D(k)}) - f^*)^2] \\
&\leq \mathbb{E}[f(x^k)] - \frac{\alpha_k^2}{2R^2(x^0)}\left(\frac{1}{\alpha_k} - \frac{1+\tau+\tau\rho^\tau}{2}\right)\mathbb{E}[(f(x^k) - f^*)^2].
\end{aligned}$$

The second and third steps are similar to the proof of Theorem 9.4.1. The last step follows from the fact that the method is a descent method in expectation. Following similar analysis as Theorem 9.4.1, we get the required result. □

9.10 Proof of Theorem 9.4.4

Proof. Let $Ax = \sum_i x_i$. Let $\tilde{x}_{k+1}$ be solution to the following optimization problem:

$$\tilde{x}^{k+1} = \arg\min_{\{x|Ax=0\}} \langle \nabla f(x^k), x - x^k\rangle + \frac{L}{2}\|x - x^k\|^2 + h(x).$$

To prove our result, we first prove few intermediate results. We say vectors $d \in \mathbb{R}^n$ and $d' \in \mathbb{R}^n$ are conformal if $d_i d_i' \geq 0$ for all $i \in [b]$. We use $d_{i_k j_k} = x^{k+1} - x^k$ and $d = \tilde{x}^{k+1} - x^k$. Our first claim is that for any d, we can always find conformal vectors whose sum is d (see [105]). More formally, we have the following result.

Lemma 9.10.1. *For any $d \in \mathbb{R}^n$ with $Ad = 0$, we have a multi-set $S = \{d_{ij}'\}_{i\neq j}$ such that d and d_{ij}' are conformal for all $i \neq j$ and $i, j \in [b]$ i.e., $\sum_{i\neq j} d_{ij}' = d$, $Ad_{ij}' = 0$ and d_{ij}' can be non-zero only in coordinates corresponding to x_i and x_j.*

Proof. We prove by an iterative construction, i.e., for every vector d such that $Ad = 0$, we construct a set $S = \{s_{ij}\}$ ($s_{ij} \in \mathbb{R}^n$) with the required properties. We start with a vector $u^0 = d$ and multi-set $S^0 = \{s_{ij}^0\}$ and $s_{ij}^0 = 0$ for all $i \neq j$ and $i, j \in [n]$. At the kth step of the construction, we will have $Au^k = 0$, $As = 0$ for all $s \in S^k$, $d = u^k + \sum_{s\in S^k} s$ and each element of s is conformal to d.

In kth iteration, pick the element with the smallest absolute value (say v) in u^{k-1}. Let us assume it corresponds to y_p^j. Now pick an element from u^{k-1} corresponding to y_q^j for $p \neq q \in [m]$ with at least absolute value v albeit with opposite sign. Note that such an element should exist since $Au^{k-1} = 0$. Let p_1 and p_2 denote the indices of these elements in u^{k-1}. Let S^k be same as S^{k-1} except for s_{pq}^k which is given by

$s_{pq}^k = s_{pq}^{k-1} + r = s_{pq}^{k-1} + u_{p_1}^{k-1}e_{p_1} - u_{p_1}^{k-1}e_{p_2}$ where e_i denotes a vector in $\mathbb{R}^n$ with zero in all components except in i^{th} position (where it is one). Note that $Ar = 0$ and r is conformal to d since it has the same sign. Let $u^{k+1} = u^k - r$. Note that $Au^{k+1} = 0$ since $Au^k = 0$ and $Ar = 0$. Also observe that $As = 0$ for all $s \in S^{k+1}$ and $u^{k+1} = \sum_{s \in S^k} s = d$.

Finally, note that each iteration the number of non-zero elements of u^k decrease by at least 1. Therefore, this algorithm terminates after a finite number of iterations. Moreover, at termination $u^k = 0$ otherwise the algorithm can always pick an element and continue with the process. This gives us the required conformal multi-set. □

Now consider a set $\{d'_{ij}\}$ which is conformal to d. We define $\hat{x}_{k+1}$ in the following manner:

$$\hat{x}_i^{k+1} = \begin{cases} x_i^k + d'_{ij} & \text{if } (i,j) = (i_k, j_k) \\ x_i^k & \text{if } (i,j) \neq (i_k, j_k) \end{cases}$$

Lemma 9.10.2. *For any $x \in \mathbb{R}^n$ and $k \geq 0$,*

$$\mathbb{E}[\|\hat{x}^{k+1} - x^k\|^2 \leq \lambda(\|\tilde{x}^{k+1} - x^k\|^2).$$

We also have

$$\mathbb{E}(h(\hat{x}^{k+1})) \leq (1-\lambda)h(x^k) + \lambda h(\tilde{x}^{k+1}).$$

Proof. We have the following bound:

$$\mathbb{E}_{i_k j_k}[\|\hat{x}^{k+1} - x^k\|^2 = \lambda \sum_{i \neq j} \|d'_{ij}\|^2 \leq \lambda \|\sum_{i \neq j} d'_{ij}\|^2 = \lambda \|d\|^2 = \lambda \|\tilde{x}^{k+1} - x^k\|^2.$$

The above statement directly follows the fact that $\{d'_{ij}\}$ is conformal to d. The remaining part directly follows from [105]. □

The remaining part essentially on similar lines as [105]. We give the details here for completeness. From Lemma 1, we have

$$\begin{aligned}
\mathbb{E}_{i_k j_k}[F(x^{k+1})] &\leq \mathbb{E}_{i_k j_k}[f(x^k) + \langle \nabla f(x^k), d_{i_k j_k} \rangle + \frac{L}{2}\|d_{i_k j_k}\|^2 + h(x^k + d_{i_k j_k})] \\
&\leq \mathbb{E}_{i_k j_k}[f(x^k) + \langle \nabla f(x^k), d'_{i_k j_k} \rangle + \frac{L}{2}\|d'_{i_k j_k}\|^2 + h(x^k + d'_{i_k j_k})] \\
&= f(x^k) + \lambda \left(\langle \nabla f(x), \sum_{i \neq j} d'_{ij} \rangle + \sum_{i \neq j} \frac{L}{2}\|d'_{ij}\|^2 + \sum_{i \neq j} h(x + d'_{ij}) \right) \\
&\leq (1-\lambda)F(x^k) + \lambda(f(x^k) + \langle \nabla f(x), d \rangle + \frac{L}{2}\|d\|^2 + h(x+d)) \\
&\leq \min_{\{y | Ay = 0\}} (1-\lambda)F(x^k) + \lambda(F(y) + \frac{L}{2}\|y - x^k\|^2) \\
&\leq \min_{\beta \in [0,1]} (1-\lambda)F(x^k) + \lambda(F(\beta x^* + (1-\beta)x^k) + \frac{\beta^2 L}{2}\|x^k - x^*\|^2) \\
&\leq (1-\lambda)F(x^k) + \lambda \left(F(x^k) - \frac{2(F(x^k) - F(x^*))^2}{LR^2(x^0)} \right).
\end{aligned}$$

The second step follows from optimality of $d_{i_k j_k}$. The fourth step follows from Lemma 9.10.2. Now using the similar recurrence relation as in Theorem 2, we get the required result. $\square$

9.11 Reduction of General Case

In this section we show how to reduce a problem with linear constraints to the form of Problem (9.4). For simplicity, we focus on smooth objective functions. However, the formulation can be extended to composite objective functions along similar lines. Consider the optimization problem

$$\min_x f(x) \text{ s.t. } Ax = \sum A_i x_i = 0,$$

where f_i is a convex function with an L-Lipschitz gradient.

Let $\bar{A}_i$ be a matrix with orthonormal columns satisfying $\text{range}(\bar{A}_i) = \ker(A_i)$, this can be obtained (e.g. using SVD). For each i, define $y_i = A_i x_i$ and assume that the rank of A_i is less than or equal to the dimensionality of x_i. [4] Then we can rewrite x as a function $h(y, z)$ satisfying

$$x_i = A_i^+ y_i + \bar{A}_i z_i,$$

for some unknown z_i, where C^+ denote the pseudo-inverse of C. The problem then becomes

$$\min_{y,z} g(y,z) \;\; \text{s.t. } \sum_{i=1}^{N} y_i = 0, \tag{9.14}$$

where

$$g(y,z) = f(\phi(y,z)) = f\left(\sum_i U_i (A_i^+ y_i + \bar{A}_i z_i)\right). \tag{9.15}$$

It is clear that the sets $S_1 = \{x | Ax = 0\}$ and $S_2 = \{\phi(y,z) | \sum_i y_i = 0\}$ are equal and hence the problem defined in (9.14) is equivalent to that in (9.1).

Note that such a transformation preserves convexity of the objective function. It is also easy to show that it preserves the block-wise Lipschitz continuity of the gradients as we prove in the following result.

Lemma 9.11.1. *Let f be a function with L_i-Lipschitz gradient w.r.t x_i. Let $g(y,z)$ be the function defined in* (9.15). *Then g satisfies the following condition*

$$\|\nabla_{y_i} g(y,z) - \nabla_{y_i} g(y',z)\| \leq \frac{L_i}{\sigma^2_{\min}(A_i)} \|y_i - y_i'\|$$
$$\|\nabla_{z_i} g(y,z) - \nabla_{z_i} g(y,z')\| \leq L_i \|z_i - z_i'\|,$$

[4]If the rank constraint is not satisfied then one solution is to use a coarser partitioning of x so that the dimensionality of x_i is large enough.

where $\sigma_{\min}(B)$ denotes the minimum non-zero singular value of B.

Proof. We have

$$\begin{aligned}\|\nabla_{y_i} g(y,z) - \nabla_{y_i} g(y',z)\| &= \|(U_i A_i^+)^\top [\nabla_x f(\phi(y,z)) - \nabla_x f(\phi(y',z))]\| \\ &\leq \|A_i^+\| \|\nabla_i f(\phi(y,z)) - \nabla_i f(\phi(y',z))\| \\ &\leq L_i \|A_i^+\| \|A_i^+(y_i - y_i')\| \leq L_i \|A_i^+\|^2 \|y_i - y_i'\| = \frac{L_i}{\sigma^2_{\min}(A_i)} \|y_i - y_i'\|,\end{aligned}$$

Similar proof holds for $\|\nabla_{z_i} g(y,z) - \nabla_{z_i} g(y,z')\|$, noting that $\|\bar{A}_i\| = 1$. □

It is worth noting that this reduction is mainly used to simplify analysis. In practice, however, we observed that an algorithm that operates directly on the original variables x_i (i.e. Algorithm 21) converges much faster and is much less sensitive to the conditioning of A_i compared to an algorithm that operates on y_i and z_i. Indeed, with appropriate step sizes, Algorithm 21 minimizes, in each step, a tighter bound on the objective function compared to the bound based (9.14) as stated in the following result.

Lemma 9.11.2. *Let g and ϕ be as defined in* (9.15). *And let*

$$d_i = A_i^+ d_{y_i} + \bar{A}_i d_{z_i}.$$

Then, for any d_i and d_j satisfying $A_i d_i + A_j d_j = 0$ and any feasible $x = \phi(y,z)$ we have

$$\begin{aligned}&\langle \nabla_i f(x), d_i \rangle + \langle \nabla_j f(x), d_j \rangle + \frac{L_i}{2\alpha}\|d_i\|^2 + \frac{L_j}{2\alpha}\|d_j\|^2 \\ &\leq \langle \nabla_{y_i} g(y,z), d_{y_i} \rangle + \langle \nabla_{z_i} g(y,z), d_{z_i} \rangle + \langle \nabla_{y_j} g(y,z), d_{y_j} \rangle + \langle \nabla_{z_j} g(y,z), d_{z_j} \rangle \\ &+ \frac{L_i}{2\alpha\sigma^2_{\min}(A_i)}\|d_{y_i}\|^2 + \frac{L_i}{2\alpha}\|d_{z_i}\|^2 + \frac{L_j}{2\alpha\sigma^2_{\min}(A_j)}\|d_{y_j}\|^2 + \frac{L_j}{2\alpha}\|d_{z_j}\|^2.\end{aligned}$$

Proof. The proof follows directly from the fact that

$$\nabla_i f(x) = A_i^{+\top} \nabla_{y_i} g(y,z) + \bar{A}_i^\top \nabla_{z_i} g(y,z).$$

□

Chapter 10

Communication-Efficient Coresets for ERM

10.1 Introduction

The primary focus of previous chapters in Part II was on asynchronous systems for solving ERM problems. In this chapter, we investigate another important consideration while solving large-scale ERM problems — communication efficiency. Recall that the key idea of ERM is to minimize the loss on the training data subject to some regularization on the model that is being learned. More formally, given the training data $P = \{(z_1, y_1), \ldots, (z_n, y_n)\}$ from a probability distribution on $\mathcal{Z} \times \mathcal{Y}$, we are interested in the following generic optimization problem:

$$\min_x f(x) \equiv \frac{\lambda}{2}\|x\|^2 + \frac{1}{n}\sum_{i=1}^{n} \ell(z_i, y_i, x) \tag{10.1}$$

Throughout this chapter we assume that ℓ is convex. Furthermore, we assume that $\mathcal{Z} = \mathbb{R}^d$ and $\mathcal{Y} = \mathbb{R}$. Note that the objective function above is strongly convex (a function f is strongly convex with modulus λ if $f(x) - \frac{\lambda}{2}\|x\|^2$ is a convex function). Problems conforming to Equation (10.1) include popular supervised learning algorithms like support vector machines and regularized logistic regression. For example, when $z_i \in \mathbb{R}^d$, $y_i \in \{-1, 1\}$ and $\ell(z_i, y_i, x) = \log(1 + \exp(-y_i x^\top z_i))$ (logistic loss), the optimization problem in Equation (10.1) corresponds to regularized logistic regression. The loss function ℓ is not necessarily smooth as in, for example, support vector machines (SVM) where $\ell(z_i, y_i, x) = \max(0, 1 - y_i x^\top z_i)$ (hinge loss).

Several algorithms have been proposed in the literature for solving optimization problems of the aforementioned form. We will briefly review a few key approaches in Section 10.1.1; however, the algorithms are either largely synchronous or communication intensive. For example, one of the popular approaches for solving such optimization problems is stochastic subgradient descent. At each iteration of the algorithm, a single training example is chosen at random and used to determine the subgradient of the objective function. While such an approach reduces the computation complexity at each iteration, the communication cost is prohibitively expensive in distributed environment.

In this chapter, we study the problem described in Equation (10.1) in the setting where the data is distributed across nodes and hence, communication is expensive in comparison to the computation time. *The main theme of this chapter is to reduce communication cost by constructing and optimizing over a small summary of the training data — which acts as a proxy for the entire data set*. Such a summary of the training points is called a *coreset*. While this methodology has been successfully applied to data clustering problems like k-means and k-median (we refer the reader to [38, 39] for a comprehensive survey), it remains largely unexplored for supervised learning and optimization problems. The goal of this chapter is to advance the frontier in this direction. In light of the above, our primary contributions in this chapter are as follows:

- We describe a general framework for designing coreset-based algorithms. This also provides insights into existing algorithms from the coresets viewpoint.
- We propose a novel coreset-based algorithm with low communication cost and provable guarantees on the convergence to the optimal solution.
- We demonstrate the efficiency of the proposed algorithm on a few real-world datasets. In particular, we show that the proposed approach reduces the communication cost significantly.

This chapter is structured as follows. We begin with a discussion on the related work in Section 10.1.1. In Section 10.2, we describe a general framework for the coreset-based methodology. We then propose a coreset-based algorithm in Section 10.3 and provide its convergence analysis. We finally conclude by demonstrating the empirical performance of the algorithm in Section 10.4.

10.1.1 Related Work

As noted earlier, problem in Equation (10.1) arises frequently in the machine learning and optimization literatures and hence has been a subject of extensive research. Consequently, we cannot hope to do full justice to all the related work. We instead mention the key relevant works here and refer the reader to the appropriate references for a more thorough coverage.

First-order methods: In large-scale machine learning and convex optimization applications, first-order methods are popular due to their cheap iteration cost. The classic approach in first order methods is the gradient descent approach. For strongly convex functions f with L-lipshictz gradient, gradient descent has linear convergence rate i.e., $f(x_t) - f(x_*) \leq \epsilon$ in $O(\log(1/\epsilon))$ iterations where x_t and x_* are the t^{th} iterate of gradient descent and the optimal solution respectively [111]. The constants can be further improved by the means of accelerating techniques [111]. On the other hand, when ℓ is non-smooth, gradient descent methods have sub-linear convergence rates.

While gradient descent methods have appealing convergence properties, they have two major shortcomings: (1) they require evaluation of n gradients at each iteration, typically leading to high computational cost, and (2) the communication costs is also high. A popular modification of this algorithm in large-scale settings is the stochastic gradient descent. While the computational cost per iteration decreases, the linear convergence property is lost. This is due to the variance introduced by stochasticity of the approach. Recently, there has been a surge in interest to address this issue by incremental methods (see [71, 153]). By reducing the variance, these approaches achieve low iteration complexity while retaining the good convergence properties. However, all these approaches still do not address the other major shortcoming — namely, high communication cost.

Active Set & Cutting plane Methods: Our approach is also related to the classic active set and cutting plane methods used in the optimization and the machine learning literatures [117]. The basic idea is to find a *working set* of constraints, i.e., those inequality constraints of the optimization problem that are either fulfilled with equality or are otherwise important to the optimization problem. These methods are particularly popular in the SVM literature. Scheinberg et al. [152] and Joachims et al. [70] provide more details on these approaches. However, these approaches are inherently sequential and not communication friendly. Moreover, it is observed that these approaches are typically outperformed by subgradient methods [159]. While the basic theme of these methods is similar to that of ours insofar that we compute a similar *summary* of the training data at each iteration, the key distinction is the approach and methodology used in constructing the summary. Moreover, our approach is much more general and can be applied to a wide range of loss functions.

Coresets: Our approach is closely related to the paradigm of coresets used in the theory literature [12, 38, 39]. The basic idea of coresets is to extract a small amount of relevant information from the given data and work on this extracted data. Coresets have been proposed on a variety of data clustering problems such as k-means, k-medians, and projective clustering. This approach is particularly important for NP-hard problems like k-means. For example, coresets of size $O(k/\epsilon^4)$ and independent of n (number of the data points) have been proposed for the k-means problem [12, 38]. If k is small, such an approach makes it possible to find optimal solution of k-means simply by an exhaustive search. Furthermore, coresets can seamlessly handle distributed and streaming settings and hence, are suitable to large-scale real-world applications. We refer the reader to the excellent (but outdated) survey on coresets [6] for more details. Recently, a unifying coreset framework has been proposed for data clustering problems [38], which provides a more comprehensive treatment; interested readers may also refer to the references therein. While there has been some progress in borrowing ideas from coresets in the context of SVMs [168], this intersection remains largely unexplored.

Distributed Methods: Owing to large-scale machine learning applications, there has been a recent surge of interest in distributed training of models. The basic idea is to solve subproblems in parallel, followed by averaging at each iteration. For example, [98, 182] propose an algorithm with a trade-off between computation and communication costs. The Alternating Direction Method of Multipliers (ADMM) [20] and its variants are also popular approaches that fall in this category. However, these strategies are either synchronous and communication unfriendly since no communication occurs

```
1: **Input:** Initial $x_0$, coefficients $\{\gamma_1, \dots, \gamma_T\}$
2: **for** t = 1 to T **do**
3:     Compute the coreset $C_{t-1}$ with the corresponding function $g_{t-1}(x; C_{t-1}, x_{t-1})$
4:     Solve the following subproblem
           $x_t = \arg\min_{x \in \Omega(x_{t-1}, R_{t-1})} g_{t-1}(x; C_{t-1}, x_{t-1})$
5:     $R_t = \gamma_{t-1} \cdot R_{t-1}$
6: **end for**
```

Algorithm 23: Generic Iterative Coreset Algorithm

during the computation phase. Mini-batch approaches have received considerable attention recently. We refer the reader to [84] and references therein for a more thorough analysis of mini-batch approaches based on stochastic gradient descent.

10.2 A General Framework

We describe our general methodology in this section. Before delving into the details of the framework, we introduce a few definitions and notations in order to simplify our exposition. We denote the objective function in Equation (10.1) by $f(x; P)$. Recall that P is the training set. The optimal solution of Equation (10.1) is denoted by x_* i.e.,

$$x_* = \arg\min_x f(x; P) \equiv \frac{\lambda}{2}\|x\|^2 + \frac{1}{n}\sum_{i=1}^{n} \ell(z_i, y_i, x).$$

We use R_* to denote $\|x_* - x_0\|$, the distance of optimal solution from the initial point. Next, we define the key ingredient in our approach.

Definition 10.2.1. *(Coreset) We call a set C an ϵ-coreset of P on a set Ω if there exists a function $g : \mathbb{R}^d \to \mathbb{R}$ such that $|g(x; C) - f(x; P)| \le \epsilon$ for all $x \in \Omega$.*

Note that the above definition is slightly different from the one typically used in the coreset literature (see [38]) in two ways: (i) the set C is not necessarily a subset of P. (ii) the coreset is restricted to the domain $x \in \Omega$. Another noteworthy point is that while coresets are classically defined as a multiplicative approximation, we use the notion of additive approximation. However, such a relaxation allows us to view other related algorithms through the lens of coresets. The key desirable property of a coreset is that the cardinality of the set C is small, which will help us reduce the overall communication complexity of the algorithm.

With this background we are ready to state our algorithm. At each iteration of the algorithm, the key component of our framework is to compute a new coreset-based on the current solution and solve the optimization problem based on that coreset. The pseudocode is given as Algorithm 23.

First, we note that algorithm is still abstract because it does not specify the method to construct the coreset C_{t-1} and the function $g_{t-1}(x; C_{t-1}, x_{t-1})$. Furthermore, feasible region of the subproblem $\Omega(x_{t-1}, R_{t-1})$ at each iteration is unspecified. These details depend on the specific coreset construction and hence, are explained during the description of the coreset. We now state a general result on performance of Algorithm 23 based on some on some important properties of the coreset.

Theorem 10.2.2. *Suppose we have the following conditions on the function g_{t-1} for $1 \le t \le T$:*

1. *g_{t-1} is an upper bound on f and is strongly convex with modulus λ'_{t-1}, for some $\lambda'_{t-1} > 0$.*
2. *The feasible region $\Omega(x_{t-1}, R_{t-1})$ is convex and contains the optimal solution x_*.*
3. *Suppose $g_{t-1}(x; C_{t-1}, x_{t-1}) \le f(x; P) + \Delta_{t-1}$ for all $x \in \Omega(x_{t-1}, R_{t-1})$ i.e., C_{t-1} is an Δ_{t-1}-coreset of P on $\Omega(x_{t-1}, R_{t-1})$.*

Then for iterates $\{x_t\}_{t=1}^T$ of Algorithm 23 we have,

$$R_t = \|x_t - x_*\| \le \sqrt{\frac{2\Delta_{t-1}}{\lambda + \lambda'_{t-1}}}.$$

Proof. For brevity, we use $f(x)$ and $g_{t-1}(x)$ to denote $f(x; P)$ and $g_{t-1}(x; C_{t-1}, x_{t-1})$ respectively. We have the following:

$$g_{t-1}(x_*) \le f(x_*) + \Delta_{t-1},$$
$$g_{t-1}(x_t) + \langle \partial g_{t-1}(x_t), x_* - x_t \rangle + \frac{\lambda'_{t-1}}{2}\|x_t - x_*\|^2 \le g_{t-1}(x_*).$$

The first inequality follows from condition 3 of the theorem. The second inequality follows from the fact that g_{t-1} is strongly convex with modulus λ'_{t-1} (condition 1). Adding the above two inequalities we get

$$g_{t-1}(x_t) + \langle \partial g_{t-1}(x_t), x_* - x_t \rangle + \frac{\lambda'_{t-1}}{2}\|x_t - x_*\|^2 \le f(x_*) + \Delta_{t-1}. \tag{10.2}$$

Because f is strongly convex with modulus λ, we have

$$f(x_*) + \langle \partial f(x_*), x_t - x_* \rangle + \frac{\lambda}{2} \cdot \|x_t - x_*\|^2 \le f(x_t).$$

Combining it with the fact that g_{t-1} is an upper bound on function f (condition 1), we have

$$f(x_*) + \langle \partial f(x_*), x_t - x_* \rangle + \frac{\lambda}{2}\|x_t - x_*\|^2 \le g_{t-1}(x_t). \tag{10.3}$$

Adding Equations (10.2) and (10.3), we get the following:

$$\langle \partial g_{t-1}(x_t), x_* - x_t \rangle + \langle \partial f(x_*), x_t - x_* \rangle + \frac{(\lambda + \lambda'_{t-1})}{2}\|x_t - x_*\|^2 \le \Delta_{t-1}. \tag{10.4}$$

To complete the proof we need the following intermediate result.

Lemma 10.2.3. *Suppose g_{t-1} satisfies the conditions in Theorem 10.2.2, then for iterates x_t, for $1 \leq t \leq T$, of Algorithm 23 we have*

$$\langle \partial g_{t-1}(x_t), x_* - x_t \rangle \geq 0 \tag{10.5}$$

$$\langle \partial f(x_*), x_t - x_* \rangle \geq 0 \tag{10.6}$$

Proof. We prove the inequality in Equation (10.5). The inequality in Equation (10.6) can be proved in a similar manner. Let $A = \Omega(x_{t-1}, R_{t-1})$ and $\mathbb{I}_A : \mathbb{R}^d \to \mathbb{R}^+$ be the indicator function corresponding to A i.e.,

$$\mathbb{I}_A(x) = \begin{cases} 0 & \text{if } x \in A \\ +\infty & \text{if } x \notin A \end{cases}$$

Recall that the x_t is the optimal solution of the following:

$$x_t = \arg\min_{x \in A} g_{t-1}(x).$$

From the optimality condition of x_t, we have $\partial g_{t-1}(x_t) + \partial \mathbb{I}_A(x_t) = 0$. Therefore, we have

$$\langle \partial g_{t-1}(x_t), x_* - x_t \rangle = \langle -\partial \mathbb{I}_A(x_t), x_* - x_t \rangle \tag{10.7}$$

Since A is convex (condition 2 of Theorem 10.2.2), the subgradient will be the normal cone of A. Using the fact that $x_*, x_t \in A$ (condtion 2 of Theorem (10.2.2)) and from the definition of the normal cone, we have

$$\langle -\partial \mathbb{I}_A(x_t), x_* - x_t \rangle \geq 0.$$

Using the above inequality in Equation (10.7), we get the required result. □

Using the inequalities from Lemma 10.2.3 in Equation (10.4) it is easy to see that the result follows. □

The above result gives an upper bound on the distance of the iterate x_t in Algorithm 23 from the optimal solution x_*. Note that the bound depends on Δ_{t-1} which in turn typically depends on the optimality of x_{t-1}. It is easy to see that convergence to the optimal solution is possible as long as $\lim_{t\to\infty} \Delta_t = 0$. It is also worth noting that result does not assume anything on the size of the coreset C_t. However, as we shall see, the communication and computation complexity of the algorithm will critically depend on $|C_t|$ at each iteration.

Before discussing our algorithm based on this framework, we consider a popular instantiation of this framework — gradient descent. For this discussion, we assume that the loss function ℓ is differentiable and has L-lipschitz gradient i.e., $\|\partial \ell(z_i, y_i, x) - \partial \ell(z_i, y_i, x')\| \leq L\|x - x'\|$. This smoothness condition on the gradient gives us the following useful result.

Lemma 10.2.4. *[111] For any function $h : \mathbb{R}^d \to \mathbb{R}$ with L-Lipschitz continuous gradient ∂h, we have*

$$h(x) \leq h(y) + \langle \partial h(y), x - y \rangle + \frac{L}{2}\|x - y\|^2, \; \forall x, y \in \mathbb{R}^d.$$

The update for gradient descent is the following:

$$x_{t+1} = x_t - \gamma \partial f(x; P) \tag{10.8}$$

where γ is the learning rate and is typically set to $1/L$. Such an update can be obtained by minimizing the upper bound on f in Lemma 10.2.4. We briefly explain how gradient descent like method fits our framework. We choose the coreset[1] $C_t = \partial f(x_t; P)$ and the function g_t as follows:

$$g_t(x; C_t, x_t) = f(x_t; P) + \langle \partial f(x_t), x - x_{t-1} \rangle + \frac{L + \lambda}{2}\|x - x_{t-1}\|^2 \tag{10.9}$$

First note that the function g_t is an upper bound of f and is strongly convex with modulus $L + \lambda$. This can be obtained from Lemma 10.2.4. Hence, g_t satisfies condition 1 of Theorem 10.2.2. Next, we set $\Omega(x_t, R_t) = \mathcal{B}(x_t, R_t)$ where $\mathcal{B}(x, R)$ represents a ball of radius R centered around x. Since this is convex and $R_t = \|x_t - x_*\|$, it is easy to see that condition 2 of Theorem 10.2.2 holds. In general, the we

Finally, g_t is an Δ_t-coreset with $\Delta_t \leq LR_{t-1}^2/2$. This can be obtained by a straightforward reasoning based on the Taylor expansion of f and Lemma 10.2.4. Thus all the conditions of Theorem 10.2.2 hold. Hence, using Theorem 10.2.2 we obtain the following corollary.

Corollary 10.2.4.1. *The iterates x_t of gradient descent algorithm like algorithm (minimizing upper bound in Equation* (10.9) *subject to the constraint on $x \in \Omega(x_t, R_t)$) satisfy*

$$R_t = \|x_t - x_*\| \leq \sqrt{\frac{2LR_{t-1}^2}{2(L + 2\lambda)}} = R_{t-1}\sqrt{\frac{1}{(1 + \frac{2\lambda}{L})}}.$$

In general, dropping the constraint that $x \in \Omega$, recovers the gradient descent algorithm. The above corollary reproduces the well-known linear convergence rate for gradient descent [111]. Note the dependence of the convergence rate on the condition number L/λ. While the result does not lead to any new convergence rates, it provides an interesting insight that gradient descent can be viewed as solving an optimization problem on a *coreset* based on the gradients at each iteration. However, it is important to note that the communication cost is still high since the gradient needs to be communicated at each iteration. Hence, gradient descent is not suitable for settings of our interest — that is, distributed settings where communication is expensive.

A natural question that arises is whether we can construct more interesting coresets than the gradients of the function. We provide an affirmative answer to this question in the next few sections.

[1]Recall that the coreset could be any summary of the data, and not necessarily one of its subsets.

10.3 Coreset Algorithm

In this section, we propose a new coreset-based algorithm. Before discussing the details of the coreset contribution, it is worth mentioning two additional assumptions; however, we should emphasize that the first assumption is only for the ease of exposition.

1. The loss function ℓ is of the form $\ell(z_i, y_i, x) = \ell(y_i x^\top z_i)$. Note the slight abuse of notation in the usage of ℓ. In what follows, the quantity $y_i x^\top z_i$ is referred to as *margin*.
2. ℓ is L-lipschitz continuous i.e., $|\ell(y_i x^\top z_i) - \ell(y_i x'^\top z_i)| \leq L|y_i x^\top z_i - y_i x'^\top z_i|$.

Loss functions that satisfy the above properties include popular choices such as logistic loss (used in logistic regression) and hinge loss (used in SVM). The significance of these assumptions will become clear as we proceed. We also need the following definitions for our discussion.

Definition 10.3.1. *(Cover) We call a set of points S as ϵ-cover of a set of points Q if for all $q \in Q$ there exists a point $s \in S$ such that $\|s - q\| \leq \epsilon$.*

Let $N(x)$ for a point x in the cover denote the set of points in Q that are closer to x than any other point in the cover S. With slight abuse of notation, let $\epsilon(Q) = [S, \beta]$, where S is an ϵ-cover of Q, and β is the vector of cardinalities of the sets $\{N(x)|x \in S\}$. Note that $\|\beta\|_1 = |Q|$.

The key insight to our coreset construction of the algorithm is that typically at each iteration there exist only a few *important* data points that are critical from the optimization perspective. For example, consider an iterative algorithm for SVMs. Intuitively, at each iteration, the points that are close to the margin are crucial in comparison to those away from the it. Furthermore, due to the piecewise linear nature of the hinge loss, the points far away from the margin can be represented by a linear function precisely. With this intuition, we now present our coreset construction.

We define the set $P' = \{z'_i\}_{i=1}^n$ where $z'_i = y_i z_i$. Our coreset construction consists of two primary steps:

Step 1: Identify points whose loss can be approximated by a linear function and construct a *single* linear function as a coreset for these points. Generally, these are points where gradient approximation is good. We denote such a function by LINEARAPPROX. The description of this function will depend on ℓ.

Step 2: Construct a cover or equivalent functional approximation for the rest of the points in the set P'. Since ℓ is assumed to be lipschitz, such a cover also provides approximation guarantees on the empirical loss on P'.

It should be emphasized that while we use the concept of cover for simplicity, a similar analysis can be carried out for clustering-based algorithms. In fact, as we will see later, all our experiments are based on clustering. At each iteration, we use disjoint sets G_t and E_t to denote the points concerned with these two steps respectively. Note that $G_t \cup E_t = P'$. We use $l_t \in \mathbb{R}^d$ to denote the linear approximation of loss for points in G_t. Our coreset is $C_t = (I_t, \beta_t, l_t)$ where $[I_t, \beta_t] = \epsilon_t(E_t)$ and function g_t in Algorithm 23 is as follows.[2]

$$g_t(x; C_t) = \frac{\lambda}{2}\|x\|^2 + \frac{1}{n}\left(\sum_{z_e \in I_t} \beta_t^e \ell(x^\top z_e) + x^\top l_t\right) + h_t \tag{10.10}$$

where $h_t = (2LR_*|E_t|\epsilon + |G_t|\delta + c)/n$ for some $\delta > 0$ and constant c. The pseudocode, based on the above key steps, is given as Algorithm 24. The size of the coreset C_t depends on the cardinality of set E_t.

```
1: **Input**: Initial $x_0$, coefficients $\{\gamma_1, \dots, \gamma_T\}$ and $\{\epsilon_1, \dots, \epsilon_T\}$
2: **for** t = 1 to T **do**
3:     $[G_{t-1}, l_{t-1}] = \textsc{LinearApprox}(P', x_{t-1}, R_{t-1})$
4:     $[I_{t-1}, \beta_{t-1}] = \epsilon_{t-1}(P' \setminus G_{t-1})$
5:     Coreset $C_{t-1} = (I_{t-1}, \beta_{t-1}, l_{t-1})$
6:     Solve the following subproblem
           $x_t = \operatorname*{arg\,min}_{x \in \Omega(x_{t-1}, R_{t-1})} g_{t-1}(x; C_{t-1})$
7:     $R_t = \gamma_{t-1} \cdot R_{t-1}$
8: **end for**
```
Algorithm 24: Iterative Coreset Algorithm

One of the key components of Algorithm 24 is the function LINEARAPPROX. As mentioned earlier, in general, this function depends on the loss function ℓ. We choose $\Omega(x_0, R_0) = \mathcal{B}(x_0, R_0)$ and $\Omega(x_t, R_t) = \Omega(x_{t-1}, R_{t-1}) \cap \mathcal{B}(x_t, R_t)$ for $t \in [1, \dots, T-1]$. Also, for the purpose of analysis, we assume the coefficients $\{\gamma_1, \dots, \gamma_T\}$ are chosen in a way such that the optimal solution lies in feasible region. We prove the following result for Algorithm 24. The proof of Theorem 10.3.2 relies on result of the generic coreset algorithm, Theorem 10.2.2.

Theorem 10.3.2. *Suppose g_t is as defined in Equation* (10.10) *and* LINEARAPPROX *satisfies the following condition:*

$$\max_{x \in \mathcal{B}(x_t, R_t)} \left| \sum_{z_g \in G_t} \ell(x^\top z_g) - [x^\top l_t + c] \right| \leq |G_t|\delta \tag{10.11}$$

where $[G_t, l_t] = \textsc{LinearApprox}(P', x_t)$ and $c \in \mathbb{R}^d$ and for all $t \in \{0, \dots, T-1\}$, then we have

$$R_{t+1} = \|x_{t+1} - x_*\| \leq \sqrt{\frac{2LR_*|E_t|\epsilon_t + |G_t|\delta}{\lambda n}}.$$

[2]Hereinafter we include the parameter x_t in the coreset description C_t.

Proof. We first observe that g_t (in Equation (10.10)) is strongly convex with modulus λ. Moreover, g_t is an upper bound on f due to the following relation:

$$\begin{aligned}
&\frac{1}{n}\left(\sum_{z_e \in I_t} \beta_t^e \ell(x^\top z_e) + x^\top l_t\right) + h_t \\
&= \frac{1}{n}\left(\sum_{z_e \in I_t} \beta_t^e \ell(x^\top z_e) + x^\top l_t + 2LR_*|E_t|\epsilon_t + |G_t|\delta + c\right) \\
&\geq \frac{1}{n}\left(\sum_{z_e \in I_t} \beta_t^e \ell(x^\top z_e) + 2LR_*|E_t|\epsilon_t + \sum_{z_g \in G_t} \ell(x^\top z_g)\right) \\
&\geq \frac{1}{n}\left(\sum_{z_p \in E_t} \ell(x^\top z_p) + \sum_{z_g \in G_t} \ell(x^\top z_g)\right) \\
&= \frac{1}{n}\sum_{z \in P'} \ell(x^\top z)
\end{aligned}$$

The first inequality follows from the definition of h_t. The second step follows from the condition on LINEARAPPROX in the theorem statement. The third step follows from the fact that ℓ is L-Lipschitz continuous and $\|\beta_{t-1}\|_1 = |E_t|$. Combining the above with regularization term proves the fact that g_t is an upper bound on f.

It is easy to see that the feasible region $\mathcal{B}(x_t, R_t)$ is convex and contains the optimal solution x_*. To obtain an upper bound on the function g_t, we observe the following:

$$\begin{aligned}
&\frac{1}{n}\left(\sum_{z_e \in I_t} \beta_t^e \ell(x^\top z_e) + x^\top l_t\right) + h_t \\
&\leq \frac{1}{n}\left(\sum_{z_e \in I_t} \beta_t^e \ell(x^\top z_e) + 2LR_*|E_t|\epsilon_t + \sum_{z_g \in G_t} \ell(x^\top z_g) + 2|G_t|\delta\right) \\
&\leq \frac{1}{n}\left(\sum_{z_p \in E_t} \ell(x^\top z_p) + 4LR_*|E_t|\epsilon_t + \sum_{z_g \in G_t} \ell(x^\top z_g) + 2|G_t|\delta\right) \\
&= \frac{1}{n}\left(\sum_{z \in P'} \ell(x^\top z) + 4LR_*|E_t|\epsilon_t + 2|G_t|\delta\right)
\end{aligned}$$

The first and second inequalities follow from the condition of the theorem statement and the Lipschitz continuous nature of the loss function ℓ.

Therefore, C_t with the corresponding function g_t is an Δ_t-coreset where $\Delta_t \leq (4LR_t|E_t|\epsilon_t + 2|G_t|\delta)/n$. The above reasoning shows that the function g_t satisfies all the conditions of Theorem 23. Applying Theorem 23 on the function g_t, we get the required result. □

It can be observed that the conditions $\epsilon_T \to 0$ and $\delta \to 0$ as $T \to \infty$ ensure convergence of the algorithm to the optimal solution. In general, we can guarantee that our solution is arbitrary close to the optimal solution by choosing δ and ϵ_t appropriately. Furthermore, we can ensure linear convergence of our algorithm by decreasing ϵ_t by a constant factor at each iteration.

It is also important to study the coreset size and the design choice of ϵ_t and δ since they determine the communication cost of our algorithm. Let $\delta = 2LR_* \min\{\epsilon_1, \ldots, \epsilon_T\}$. For this value of δ, we observe the following.

1. The size of the coreset depends on the cardinality of G_t. In general, larger the cardinality of G_t, smaller is the size of the coreset. Furthermore, as a general rule of thumb, if ℓ is asymptotically linear i.e., $\lim_{|m| \to \infty} |\ell(m) - (cm + d)| = 0$ for some constants c, d, the performance of our algorithm will depend on the rate of asymptotic linearity.
2. We typically require $\epsilon_t \leq \epsilon_{t+1}$ for all $t \in \{0, \ldots, T-1\}$. With such a choice, if $|G_t|$ does not decrease, size of the coreset may increase. However, observe that R_t decreases. Thus, typically more points satisfy Equation (10.11), and the cardinality of G_t usually decreases.
3. Suppose a subset of P' satisfies Equation (10.11) at iteration t then it will always satisfy the condition in future iterations. This is due to the fact that feasible region shrinks at each iteration. Hence, size of the coreset is always non-increasing.

While the above remarks provide informal reasoning for the size of the coreset, it does not provide a formal analysis. In order to gain a better understanding, we discuss the implementation of this algorithm and provide a more formal analysis in the case of logistic regression and SVMs. To this end, let us first discuss the function LINEARAPPROX for specific cases.

LINEARAPPROX for differentiable loss functions: The linear approximation in the differentiable case can be obtained through the first-order Taylor expansion of the loss function. More formally, we have

$$\ell(x^\top z_k) = \ell(x_t^\top z_k) + \partial\ell(x_t^\top z_k)(x^\top z_k - x_t^\top z_k) + \frac{\partial^2 \ell(\zeta)}{2}(x^\top z_k - x_t^\top z_k)^2$$

for some $\zeta = \tilde{x}_t^\top z$ where $\|\tilde{x}_t - x_t\| \leq R_t$ since out feasible region satisfies $\|x - x_t\| \leq R_t$. The key step is to bound the term $\partial^2 l(\zeta)$. This bound will depend on the structure of the loss function. We now derive these bounds for logistic regression. We want the following to ensure the condition in Theorem 10.3.2 with $l_t = \sum_{z_k \in G_t} \partial\ell(x_t^\top z_k) z_k$ and $c = \sum_{z_k \in G_t} [\ell(x_t^\top z_k) - \partial\ell(x_t^\top z_k) x_t^\top z_k]$:

$$\frac{\partial^2 l(\zeta)}{2}(x^\top z_k - x_t^\top z_k)^2 \leq \delta$$

for all $x_k \in G_t$. The above statement is true when

$$\frac{\partial^2 l(\zeta)}{2} R_t^2 \|z_k\|^2 \leq \delta \tag{10.12}$$

This can be obtained by a straightforward application of the Cauchy-Schwartz inequality. The final step is to derive an upper bound on $\partial^2 l(\zeta)$. For logistic loss we have

$$\partial^2 l(\zeta) = \frac{1}{(1+\exp(-\zeta))(1+\exp(\zeta))}.$$

Without loss of generality, we can assume $\zeta > 0$. Then we have $\partial^2 l(\zeta) \leq 1/(2(1+\exp(\zeta)))$. Using the above inequality it is easy to see that Equation (10.12) is satisfied if

$$\frac{R_t^2 \|z_k\|^2}{2(1+\exp(\zeta))} \leq \delta.$$

We observe that

$$\begin{aligned}\frac{R_t^2 \|z_k\|^2}{2(1+\exp(\zeta))} &= \frac{R_t^2 \|z_k\|^2}{2(1+\exp(x_t^\top z_k + \zeta - x_t^\top z_k))} \\ &\leq \frac{R_t^2 \|z_k\|^2}{2(1+\exp(x_t^\top z_k)\exp(-R_t\|z_k\|)}\end{aligned}$$

This follows from the fact that $\zeta = \tilde{w}_t^\top x$ where $\|\tilde{x}_t - x_t\| \leq R_t$. Hence, for logistic regression, the goal of LINEARAPPROX is to identify all points satisfying

$$V_t(z_k) = \frac{R_t^2 \|z_k\|^2}{2(1+\exp(x_t^\top z_k)\exp(-R_t\|z_k\|)} \leq \delta$$

and place these points in the set G_t. The linear function to be used for approximation is obtained from the first-order Taylor expansion. The pseudocode for LINEARAPPROX in case of logistic regression is given in Algorithm 25.

```
1: Input: $P', x_t, R_t$
2: $G_t = \{z_k \in P' \mid V_t(z_k) \leq \delta\}$
3: $l_t = -\sum_{z_k \in G_t} \frac{z_k}{(1+\exp(x_t^\top z_k))}$
```

Algorithm 25: LINEARAPPROX for Logistic Regression

Furthermore, we can also obtain a relationship between the margin of z_k with respect to the optimal solution and the iteration at which the point z_k moves to the set G_t. We note the following:

$$\begin{aligned}\frac{R_t^2 \|z_k\|^2}{2(1+\exp(\zeta))} &= \frac{R_t^2 \|z_k\|^2}{2(1+\exp(x_*^\top z_k + \zeta - x_*^\top z_k))} \\ &\leq \frac{R_t^2 \|z_k\|^2}{2(1+\exp(M_k^*)\exp(-2R_t\|z_k\|)} \\ &\leq \frac{R_t^2 \|z_k\|^2 \exp(2R_t\|z_k\|)}{2\exp(M_k^*)} \leq \frac{\exp(3R_t\|z_k\|)}{2\exp(M_k^*)}\end{aligned}$$

where $M_k^* = |x_*^\top z_k|$. The first step follows from triangle inequality and the fact that $\zeta = \tilde{x}_t^\top z$ where $\|\tilde{x}_t - x_t\| \leq R_t$ and $\|\tilde{x}_* - x_t\| \leq R_t$. The final step follows from the fact that $z^2 \leq \exp(z)$ for $z \geq 0$. Therefore, from above inequality it is easy to see that Equation (10.12) is satisfied when the following holds

$$R_t \leq \max\left\{\frac{\sqrt{\delta}}{\|z_k\|}, \frac{M_k^* + \log(2\delta)}{3\|z_k\|}\right\}. \tag{10.13}$$

LINEARAPPROX for SVM: For SVM, the implementation of LINEARAPPROX is pretty straightforward. Due to the piecewise linear nature of the hinge loss, the condition in Equation (10.10) is satisfied with $\delta = 0$ if $x^\top z_k$ is greater than 1 (or less than 1) for the whole feasible region $\|x - x_t\| \leq R_t$. This is satisfied when

$$R_t \leq \frac{|1 - x_t^\top z_k|}{\|z_k\|}$$

The pseudocode for LINEARAPPROX in case of SVM is given in Algorithm 26.

```
1: Input: $P', x_t, R_t$
2: $G_t = \{z_k \in P' \,|R_t \leq |1 - x_t^\top z_k| / \|z_k\|\}$
3: $l_t = -\sum_{z_k \in G_t} \mathbb{1}(1 - x_t^\top z_k < 0) z_k$
```

Algorithm 26: LINEARAPPROX for SVM

Similarly to the case of logistic regression, we analyze how the margin of z_k affects when it get included in the set G_t. For this, note the following:

$$1 - x_t^\top z_k = 1 - x_*^\top z_k - (x_t - x_*)^\top z_k$$

Again, the above quantity will not change sign when $x^\top z_k$ is greater than 1 (or less than 1) for the whole feasible region $\|x - x_t\| \leq R_t$. Based on the expression above, this is satisfied when

$$\|x_t - x_*\| \|z_k\| \leq |1 - x_*^\top z_k| = M_k^*$$

It is obtained by application of the Cauchy-Schwartz inequality. Note the difference in the definition of M_k^* in comparison to logistic regression. Hence, condition in Equation (10.10) will be satisfied when

$$R_t \leq \frac{M_k^*}{\|z_k\|} \tag{10.14}$$

Let us make a final remark before proceeding to the experimental section. It should be emphasized that based on Equations (10.13) and (10.14), the cardinality of G_t critically depends on the margin of the training points. If the margin of the training points is large, then the coreset size is small and consequently the communication and computation costs are low. Hence, our algorithm is naturally adaptive to the hardness of the optimization problem.

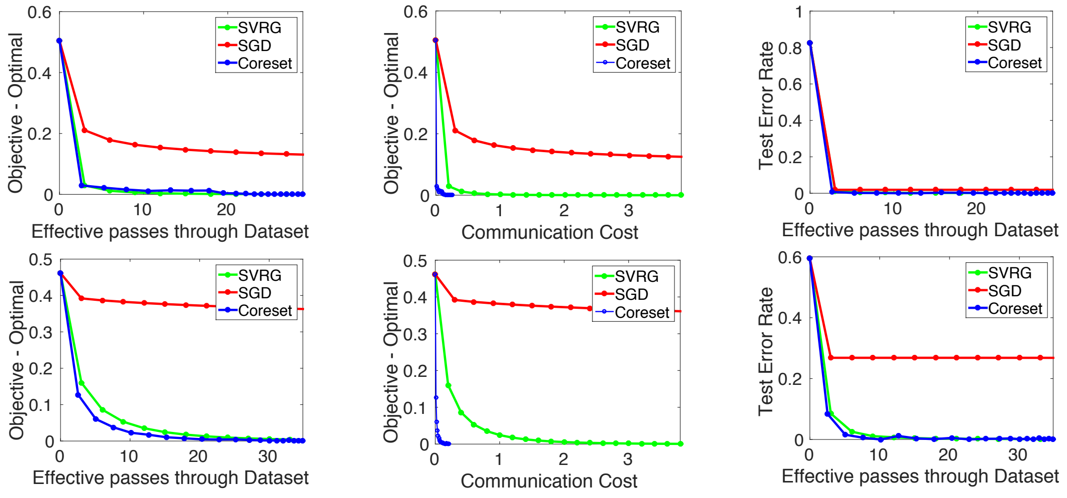


Figure 10.1: l_2-regularized logistic regression on ijcnn1 (top) and cod-rna (bottom) datasets. We compare our algorithm with mini-batch SVRG and SGD. Training loss residual is shown with respect to passes through the dataset and communication cost (left and central columns). Test error with respect to the passes through the dataset is shown in the right column.

10.4 Experiments

We present our empirical results in this section. To evaluate the performance of our algorithm, we focus on the task of regularized logistic regression. Recall that Problem (10.1) in this case is of the following form:

$$\min_x f(x) \equiv \frac{\lambda}{2}\|x\|^2 + \frac{1}{n}\sum_{i=1}^{n}\log(1+\exp(-y_i x^\top z_i)).$$

We use a simulated Matlab implementation for all the algorithms reported in this section. We used the following datasets for our experiments.

Dataset	# examples	# features
ijcnn1	49,990	22
cod-rna	59,535	8
w8a	64,700	300
covertype	581,012	54

All these datasets can be downloaded from the LIBSVM website [3]. Similar to [71], each of these datasets are scaled to $[-1,1]$. We split each of these datasets in 3:1 ratio for training and testing purposes respectively.

The regularization parameter λ in Problem (10.1) is $1/n$. Recall n is the size of the training set. Note that such a choice results in high condition number and consequently increases the difficulty of the problem. All the experiments were conducted by fixing

[3]http://www.csie.ntu.edu.tw/ cjlin/libsvmtools/datasets/

10 different random seeds for each dataset and results are reported by averaging over these 10 runs.

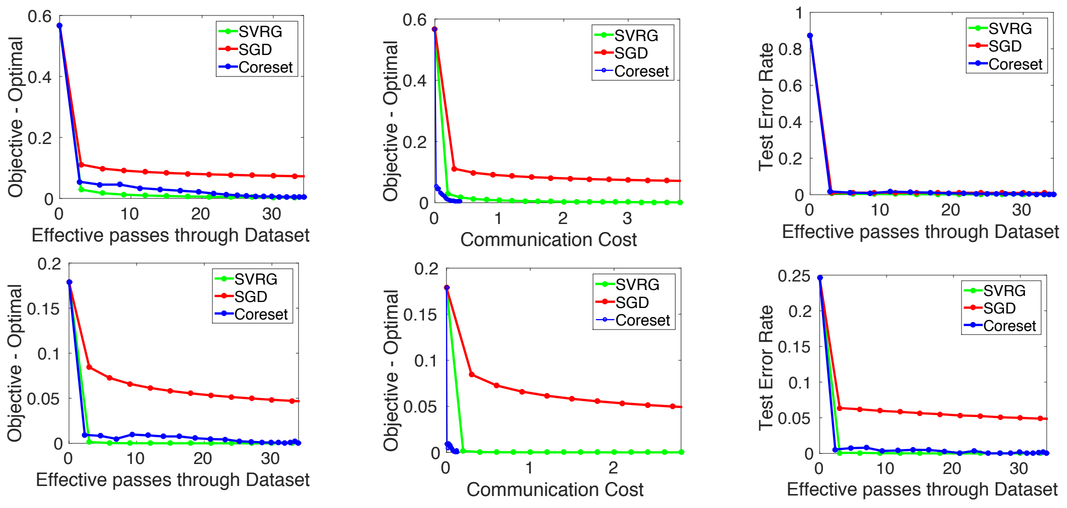


Figure 10.2: l_2-regularized logistic regression on more datasets w8a (top) and covertype (bottom). Similar to the previous case, we compare our algorithm with mini-batch SVRG and SGD.

We use PROXSVRG (see [174]) for solving the subproblems of Algorithm 24 at each iteration. SVRG is a incremental first-order method that can be used for solving optimization problems of form (10.1). The origin is used as the initial point for the for all our experiments. The number of inner iterations is set to $m = 2n$ for the SVRG algorithm. The step size parameter for each dataset is chosen so as to give the fastest convergence for SVRG.

For our experiments, we choose $\gamma_1 = \gamma_2 = \cdots = \gamma_T = \gamma$ in Algorithm 24 where γ is such that $R_T = 0.01$. Such a choice is reasonable in the case of linearly convergent algorithms — which is the scenario we anticipate for our algorithm. As mentioned earlier, we use a clustering based algorithm instead of the cover. The main rationale behind such a choice is the availability of coresets for data clustering problems. We use coresets for k-means clustering for Algorithm 24. The sensitivity based coreset for k-means is used in all our experiments. We refer interested reader to [38, 39] for more details of the coreset. For all datasets except covertype, we set the coreset size to be 500. This value is set to 2000 in case of covertype. Note that these coreset sizes are much smaller in comparison to the training data.

We compare our algorithm with SVRG (see [71]) and SGD. A mini-batch version of these methods is used in order to reduce the communication cost of these approaches. We use a mini-batch size $b = 10$ in all our experiments. The number of inner iterations in SVRG is $m = \lceil \frac{2n}{b} \rceil$ in all our experiments in order to limit the total inner iterations to the recommended $2n$ iterations. For SGD, we use the learning rate of $\alpha/\sqrt{t}$ where α is the step size used for the all the algorithms for that dataset.

We report the training loss residual i.e., objective of Problem (10.1) minus optimal (obtained by running gradient descent for very long time) and test error rate of the algorithms with respect to the number of effective passes through the dataset i.e., ratio of the number of gradients evaluated to the size of the training data. This provides information about the computation complexity of the algorithm. To measure the communication cost of the algorithm, we similarly report the effective communication of datasets i.e., ratio of the number of d dimensional vectors communicated to the size of the training data.

Figures 10.1 and 10.2 show the performance of the algorithms on the aforementioned datasets. We have several observations from these empirical results. First, we observe that SVRG outperforms SGD in terms all the metrics of our interest. This observation is not surprising given the linear convergence of SVRG in comparison to the sublinear convergence of SGD. We then observe that our algorithm is competitive to SVRG in terms of training loss residual and test error rate (show in first and third columns of the figures respectively). However, our major gain is in the communication cost of the algorithm (shown in central column of the figures). As seen in the these figures, our algorithm performs much better in comparison to other algorithms in terms of communication cost. In other words, for the same communication cost, our algorithm has much lower objective value in comparison to SVRG and SGD. We believe the performance of our algorithm can be further improved by utilizing the coresets of the previous iteration and is part of our ongoing investigation.

10.5 Discussion

This chapter introduces a novel general strategy for designing communication efficient empirical risk minimization algorithms. The key to our approach is the concept of coreset — the idea of constructing small summary of training data and optimizing over this summary. We presented convergence analysis for the algorithm. While we illustrated this strategy on two popular supervised learning problems — logistic regression and support vector machines, our methodology is general and is applicable to a much wider setting. Furthermore, our empirical results demonstrate that the algorithm is computational and communicational efficient. In the next chapter, we present an alternate paradigm for reducing communication complexity in distributed optimization setting.

Chapter 11

Communication-Efficient Distributed Optimization for ERM

11.1 Introduction

In this chapter, we explore an alternate approach for reducing communication overhead in distributed machine learning systems designed for ERM problems. With the advent of large scale datasets, distributed machine learning has garnered significant attention recently. For example, large scale distributed machine learning systems such as the *Parameter server* [84], *GraphLab* [176] and *TensorFlow* [1] work with datasets sizes in the order of hundreds of terabytes. When dealing with datasets of such scale in distributed systems, computational and communication workloads need to be designed carefully. This in turn places primary importance to computational and communication resource constraints on the algorithms used in these machine learning systems.

Here, we study the problem of distributed optimization for solving *empirical risk minimization* problems, where one seeks to minimize average of N functions $\min_{x \in \mathbb{R}^d} f(x) := \frac{1}{N}\sum_{i=1}^{N} f_i(x)$. Many problems in machine learning, such as logistic regression or deep learning, fall into this setting. Formally, we assume a distributed system consisting of K machines, where machine k has access to a (nonempty) subset of the data indexed by $\mathcal{P}_k \subset [N] := \{1, \dots, N\}$. We assume that $\{\mathcal{P}_k\}$ forms a partition of $[N]$. Our goal can then thus be reformulated as

$$\min_{x \in \mathbb{R}^d} f(x) = \frac{1}{K}\sum_{k=1}^{K} F_k(x), \text{ where } F_k(x) := \frac{K}{N}\sum_{i \in \mathcal{P}_k} f_i(x). \tag{11.1}$$

We assume that (11.1) has an optimal solution. By $\hat{x}$ we denote an arbitrary optimal solution: $\hat{x} \in \arg\min_x f(x)$. The fundamental constraint is that machine k can directly access only (the data describing) function $F_k(x)$, i.e., functions $f_i(x)$ for $i \in \mathcal{P}_k$. Typically, $f_i(x) = \phi_i(x^\top z_i)$, where z_i is the i^{th} data point, and $\phi_i(\cdot)$ is a loss function (dependence on i indicates potential dependence on a label). The function f_i is assumed to be continuous but *not* necessarily convex — the structure also encompasses problems in deep learning.

The basic benchmark algorithm for solving the optimization problem is gradient descent (GD). Each iteration of GD performs the update step $x \leftarrow x - \frac{h}{K}\sum_{k=1}^{K} \nabla F_k(x)$, where $h > 0$ is a stepsize parameter. In a distributed system, this amounts to calculation of the gradient $\nabla F_k(x)$ by each machine k and then sending an update to a central node to compute the full update direction $\frac{1}{K}\sum_{k=1}^{K} \nabla F_k(x)$. Although such an update sequence is simple, it is often communication intensive due to the communication involved at each iteration of the algorithm. Furthermore, because of the communication delay, computational resources at each machine are often under-utilized. While the latter problem can be addressed by using asynchronous and stochastic variants of gradient descent [128, 133], these algorithms still incur high communication costs at each iteration.

Recently, there have been considerable theoretical advances in our understanding of the communication overheads in solving (11.1). In particular, three different methods address this issue by designing a procedure that uses significant amount of computation locally, between rounds of communication. These methods are DANE [163], DISCO [180] and COCOA+ [67, 95, 96, 177]. Both DANE and DISCO show that when the functions $\{F_k\}_{k=1}^{K}$ in (11.1) are similar (see Definition 11.3.2) or exhibit special structure, one could obtain the optimal solution in considerably fewer rounds of communication.

At the t^{th} iteration of DANE, the following general subproblem has to be solved *exactly* by machine k, with $\eta \geq 0$ and $\mu \geq 0$ being parameters of the method:

$$\min_{x \in \mathbb{R}^d} F_k(x) - \left\langle \nabla F_k(x^{t-1}) - \eta \nabla f(x^{t-1}), x \right\rangle + \frac{\mu}{2}\|x - x^{t-1}\|^2,$$

where $\langle \cdot, \cdot \rangle$ is the standard inner product and $\|\cdot\| := \langle \cdot, \cdot \rangle^{1/2}$ is the standard Euclidean norm. The gradient $\nabla f(x^{t-1})$ is computed in a distributed fashion at the start, followed by aggregation of individual updates. DISCO uses an *inexact* damped Newton method for solving the problem (11.1) [180]; and hence is a *second* order method. Thus, both DANE and DISCO require strong oracle access — either in the form of oracle to solve another optimization problem exactly, or second order information. This is in sharp contrast with COCOA+, where (similarly to DANE) a general local problem is formulated, but is needed to be solved only approximately. This enables use of essentially any algorithm to be used locally for a number of iterations, making the method much more versatile.

A natural idea is to solve DANE subproblem approximately using a first-order method. However, from the point of analysis, this leads to an additional error. Hence, it is not clear whether such method enjoys any theoretical guarantees, as such a generalization is non-trivial. Based on this intuition, we develop an inexact version of DANE (refereed to as INEXACTDANE) and provide its convergence analysis. We demonstrate that our proposed approach is significantly more robust to 'bad' data partitioning, and also highlight a connection to a distributed version of SVRG algorithm [71, 74]; thus, yielding partial convergence guarantees for a method that has been observed to perform well in practice [76], but has not been successfully analyzed.

While INEXACTDANE is practically appealing in comparison to DISCO due to its simplicity and possible reliance only on first-order information, DISCO is *theoretically* superior to InexactDane in terms of communication complexity, as it nearly matches the lower bounds derived in [175]. To address this issue, we build upon InexactDane and propose an approach, referred to as Aide, that matches (up to logarithmic factors) these communication complexity lower bounds and can be implemented using a first-order oracle. To our knowledge, ours is the first work that can be implemented using a first-order oracle, but at the same time achieves near optimal communication complexity for the setting considered in this chapter.

11.1.1 Related Work

The literature on distributed optimization is vast and hence, we restrict our attention to the works most relevant to this chapter. As mentioned earlier, the most straightforward approach for solving the distributed optimization is using gradient descent or its accelerated version. When the function f is L-smooth and λ-strongly convex, the communication complexity of accelerated gradient descent is $O(\sqrt{L/\lambda}\log(1/\epsilon))$ for achieving ϵ-suboptimal solution [111]. Another possible approach, often referred to as "one shot averaging", is solve the local optimization problem parallelly at each machine and then compute average of the solutions [181, 182]. However, it can be shown that "one shot averaging" can produce significantly worse solution than the optimal solution $\hat{x}$.

ADMM (alternating direction method of multipliers) is another popular approach for solving distribution optimization problems. Under certain conditions, ADMM is shown to achieve communication complexity of $O(\sqrt{L/\lambda}\log(1/\epsilon))$ for L-smooth and λ-strongly convex function. More recently, the above mentioned Dane, Disco and Cocoa+ algorithms have been proposed to tackle the problem of reducing the communication complexity in solving problems of form (11.1). We defer a detailed comparison of our algorithms with Dane and Cocoa+ to Section 11.7 and Appendix 11.13. The lower bounds on the communication complexity for solving problems of form (11.1) have been obtained in [175] (see Section 11.8 for more details). We refer the readers to [163, 175] for a more comprehensive coverage of the related work.

11.2 Algorithm: InexactDane

The Dane algorithm [163] proceeds as follows. At iteration t, node k *exactly* solves the subproblem $\hat{x}_k^t = \arg\min_x g_{k,\mu}^t(x)$, where

$$g_{k,\mu}^t(x) := F_k(x) - \left\langle \nabla F_k(x^{t-1}) - \eta \nabla f(x^{t-1}), x \right\rangle + \tfrac{\mu}{2}\|x - x^{t-1}\|^2, \qquad (11.2)$$

and $\mu, \eta \geq 0$ are parameters. After this, the method computes $x^t = \sum_{k=1}^K \hat{x}_k^t / K$, which involves communication, and the process is repeated. Our first method, InexactDane , modifies this algorithm by allowing the subproblems to be solved inexactly. In particular, let x_k^t denote the point obtained by minimizing $g_{k,\mu}^t$ only approximately (for example, using Quartz [125] or Svrg [71, 74]). The pseudocode of this method is formalized

Algorithm 27: INEXACTDANE (f, w^0, s, γ, μ)

```
Data: $f(x) = \frac{1}{K}\sum_{k=1}^K F_k(x)$, initial point $x^0 \in \mathbb{R}^d$, inexactness parameter $0 \le \gamma < 1$
for $t = 1$ to $s$ do
    for $k = 1$ to $K$ do in parallel
        Find an approximate solution $x_k^t \approx \hat{x}_k^t := \arg\min_{x \in \mathbb{R}^d} g_{k,\mu}^t(x)$
        Option I: $\|x_k^t - \hat{x}_k^t\| \le \gamma \|x^{t-1} - \hat{x}_k^t\|$
        Option II: $\|\nabla g_{k,\mu}^t(x_k^t)\| \le \gamma \|\nabla g_{k,\mu}^t(x^{t-1})\|$ and $\|x_k^t - \hat{x}_k^t\| \le \gamma \|x^{t-1} - \hat{x}_k^t\|$
    end
    $x^t = \frac{1}{K}\sum_{k=1}^K x_k^t$
end
return $w^t$
```

in Algorithm 27. The parameter γ refers to the *level of inexactness* allowed (small γ means higher accuracy). Note that each iteration of INEXACTDANE can be solved in an embarrassingly parallel fashion.

The communication complexity of an algorithm is defined as the number of communication rounds required by the algorithm to reach a solution x^t satisfying specific convergence criteria such as $f(x^t) - f(\hat{x}) \le \epsilon$. In each communication round, the machines can only send information linear in size of the dimension d. This is also the notion of communication round used in [175].

We start with the analysis of *quadratic case* (Section 11.3), after which we deal with strongly convex, weakly convex and nonconvex cases (Section 11.4).

11.3 Analysis of INEXACTDANE: Quadratic Case

In this section we assume that $F_k(x) = \frac{1}{2}x^\top H_k x + l_k^\top x$, where $H_k \in \mathbb{R}^{d\times d}$ and $l_k \in \mathbb{R}^d$ (this is the case when the functions $\{f_i\}_{i=1}^N$ in (11.1) are quadratic). We write $H := \frac{1}{K}\sum_{k=1}^K H_k$ and $l := \frac{1}{K}\sum_{k=1}^K l_k$. For a square matrix A, by $\|A\|$ we denote its spectral norm. Our first result plays a central role in the analysis of quadratic functions. Proofs are provided in the Appendix.

Theorem 11.3.1. *Assume $H \succ 0$. Let $\mu \ge 0$ and define $\tilde{H}^{-1} := \frac{1}{K}\sum_{k=1}^K (H_k + \mu I)^{-1}$. Further, choose $0 \le \gamma < 1$, $\eta > 0$ and define*

$$\rho := \|\eta \tilde{H}^{-1} H - I\| + \frac{\eta\gamma}{K}\sum_{k=1}^K \|(H_k + \mu I)^{-1} H\|.$$

Let $\{x^t\}$ be the iterates of INEXACTDANE *(Algorithm 27 applied with Option I). Then for all $t \ge 1$ we have $\|x^t - \hat{x}\| \le \rho \|x^{t-1} - \hat{x}\|$.*

Suppose, $0 < \lambda \preceq H_k \preceq L$ for all $k \in [K]$, then with $\eta = 1$, sufficiently large μ and small γ, one can ensure that $\rho < 1$. We are interested in the setting where the functions $\{F_k\}$ are "similar". Following [175], we shall measure similarity via the notion of δ-*relatedness*, defined next.

Definition 11.3.2. *We say that quadratic functions $\{F_k\}$ are δ-related, if for all $k, k' \in [K]$ we have*

$$\|H_k - H_{k'}\| \leq \delta, \qquad \|l_k - l_{k'}\| \leq \delta.$$

The following theorem specifies the convergence rate in the case of δ-related objectives. We are particularly interested in the setting where δ is small, which is the case that should allow stronger communication complexity guarantees [163, 175].

Theorem 11.3.3. *Let $0 < \lambda \leq L$ be such that $\lambda I \preceq H \preceq LI$, and assume that $\{F_k\}$ are δ-related, with $\delta \geq 0$. Choose $\mu = \max\{0, (8\delta^2/\lambda) - \lambda\}$, $\eta = 1$ and let*

$$\gamma = \begin{cases} 1/8 & \text{if } 2\sqrt{2}\delta \leq \lambda \\ \lambda^2/(192\delta^2) & \text{otherwise} \end{cases} \qquad \tilde{\rho} := \begin{cases} 2/3 & \text{if } 2\sqrt{2}\delta \leq \lambda \\ 1 - (\lambda^2/24\delta^2) & \text{otherwise} \end{cases}$$

The iterates $\{x^t\}$ of InexactDane *(Algorithm 27 with Option I) satisfy $\|x^t - \hat{x}\| \leq \tilde{\rho}\|x^{t-1} - \hat{x}\|$.*

It is interesting to note that solving the local subproblems beyond the accuracy γ stated in the above result does *not* lead to a better overall complexity result. That is, the required accuracy at the nodes of the distributed system should not be perfect, but should instead depend in δ and λ. We have the following corollary, translating the result into a bound on the number of iterations (i.e., number of communication rounds) sufficient to obtain an ϵ-solution.

Corollary 11.3.3.1. *Let the assumptions of Theorem 11.3.3 be satisfied, and pick $0 < \epsilon < L\|x^0 - \hat{x}\|^2$. If*

$$t = \tilde{O}\left(\frac{\delta^2}{\lambda^2}\log\left(\frac{L\|x^0 - \hat{x}\|^2}{\epsilon}\right)\right),$$

then $f(x^t) \leq f(\hat{x}) + \epsilon$.

The $\tilde{O}(\cdot)$ notation hides few logarithmic factors. In the situation when for all k, H_k arises as the average of Hessians of n i.i.d. quadratic functions with eigenvalues upper-bounded by L, it can be shown that $\delta = O(L/\sqrt{n})$, hence $t = \tilde{O}((L/\lambda)^2/n \log(1/\epsilon))$ (see Corollary 11.9.2.1 in the Appendix). This arises for instance in a linear regression setting where the samples at each machine are i.i.d — a scenario typically assumed in distributed machine learning systems. Note that, in this case, the communication complexity *decreases* as n increases.

11.4 Analysis of InexactDane: General Case

In this section, we present the results for general strongly convex case, weakly convex case, and non-convex case. Proofs are provided in the Appendix.

A function $\psi : \mathbb{R}^d \to \mathbb{R}$ is called *L-smooth* if it is differentiable, and if for all $x, y \in \mathbb{R}^d$ we have $\|\nabla\psi(x) - \nabla\psi(y)\| \leq L\|x - y\|$. It is called *$\lambda$-strongly convex* if $\psi(x) \geq \psi(y) + \langle\nabla\psi(y), x - y\rangle + \frac{\lambda}{2}\|x - y\|^2$, for all $x, y \in \mathbb{R}^d$ (if ψ is twice differentiable, this is equivalent to requiring that $\nabla^2\psi(x) \succeq \lambda I$ for all $x \in \mathbb{R}^d$). If $\lambda > 0$, we say that ψ is strongly convex. In this case, $\kappa = L/\lambda$ denotes the condition number. We say that ψ is *weakly convex* if it is 0-strongly convex.

11.4.1 Strongly convex case

Our analysis of the strongly convex case follows along the lines of [163] and incorporates the inexactness in solving the subproblem in (11.2) at each iteration.

Theorem 11.4.1. *Assume that for all $k \in [K]$, F_k is L-smooth and λ-strongly convex, with $\lambda > 0$. Let*

$$\tilde{\rho} := \left[\frac{(1-\gamma)^2}{\eta(L+\mu)} - \frac{2L}{(\lambda+\mu)^2} - \frac{2\gamma(L+\mu)}{\eta(\lambda+\mu)^2} \right] \eta^2 \lambda. \tag{11.3}$$

Suppose $\eta > 0$, $0 \le \gamma < 1$ and $\mu > 0$ are chosen such that $0 < \tilde{\rho} < 1$. Then for the iterates of InexactDane *(Algorithm 27 with Option II) we have: $f(x^t) - f(\hat{x}) \le (1 - \tilde{\rho}) \left(f(x^{t-1}) - f(\hat{x}) \right)$.*

By setting $\gamma = 0$, we roughly recover the corresponding result in [163] covering the exact case. Note that γ only has a weak effect on the convergence rate. For instance, with $\gamma = 1/8$, $\eta = 1$ and $\mu = 6L - \lambda$, we require $O(\frac{L}{\lambda} \log(1/\epsilon))$ iterations to achieve a solution x^t such that $f(x^t) - f(\hat{x}) \le \epsilon$.

11.4.2 Weakly convex case

We analyze the weakly convex case by using a perturbation argument. This allows us to use the analysis of strongly convex case for proving the convergence analysis. In particular, we consider the following function $f_\epsilon(x) = f(x) + \frac{\epsilon}{2}\|x - x^0\|^2$, which is essentially a perturbation of the function f. First, note that if f is L-smooth then f_ϵ is $(L+\epsilon)$-smooth and ϵ-strongly convex. Applying InexactDane (Algorithm 27) on the function f_ϵ and using Theorem 11.4.1, we get the following:

$$f_\epsilon(x^t) - \min_x f_\epsilon(x) \le (1 - \rho_\epsilon)^s \left[f_\epsilon(x^0) - \min_x f_\epsilon(x) \right] \tag{11.4}$$

where ρ_ϵ is defined as in (11.3) with λ and L replaced by ϵ and $L + \epsilon$ respectively. Using the above relationship, the corollary below follows immediately.

Corollary 11.4.1.1. *Suppose the function f is weakly convex. Then the iterates of* InexactDane *in Algorithm 27 (Option II) with $\eta = 1$, $\gamma = 1/8$ and $\mu = 6L + 5\epsilon$ applied to f_ϵ after $s = \tilde{O}(L \log(1/\epsilon)/\epsilon)$ iterations satisfy $f(x^t) \le f(\hat{x}) + O(\epsilon)$.*

The result essentially shows sublinear convergence rate (up to logarithmic factor) of InexactDane for weakly convex functions and weak dependence on the inexact parameter γ.

11.4.3 Nonconvex case

Finally, we look at the case where function f can be nonconvex. The key observation is that by choosing large enough μ one can have a strongly convex $g^t_{k,\mu}$ in (11.2). The existence of such a μ is due to the Lipschitz continuous nature of gradient of f. This allows us to solve the subproblem efficiently and makes it amenable to analysis. For the purpose of theoretical analysis, in the nonconvex case, we select x^t arbitrarily from $\{x_k^t\}_{k=1}^K$ instead of averaging as in Algorithm 27.

Theorem 11.4.2. *Assume that for all $k \in [K]$, F_k is L-smooth. Suppose the iterates x_k^t of Algorithm 27 (with Option II) for some $0 \leq \gamma < 1$. Furthermore, let*

$$\theta := \left[\frac{(1-\gamma)^2}{\eta(L+\mu)} - \frac{2L}{(\mu-L)^2} - \frac{2\gamma(L+\mu)}{\eta(\mu-L)^2}\right]\eta^2,$$

where $\eta > 0$, $0 \leq \gamma < 1$ and $\mu > L$ are such that $\theta > 0$. Then, we have

$$\min_{0 \leq t' \leq t-1} \|\nabla f(x^{t'})\|^2 \leq \frac{f(x^0) - f(\hat{x})}{\theta t}.$$

With $\eta = 1$, one could choose $\mu = 10L$ and constant inexactness of $\gamma = 1/8$ to get $\theta \geq 1/100L$. Again, similar to the strongly convex case, we see a very weak effect of γ on the convergence rate. In other words, with constant approximation at each local node, we can obtain $O(1/t)$ convergence rate to a stationary point for nonconvex functions.

11.5 Accelerated Distributed Optimization

In this section, we study an accelerated version of InexactDane algorithm. While InexactDane algorithm reduces the communication complexity in distributed settings, it is known that it is *not* optimal i.e., there is a gap between upper bounds of InexactDane and the lower bounds for the communication complexity proved in [175]. To this end, we propose an accelerated variant of InexactDane algorithm and prove that it matches the lower bounds (upto logarithmic factors) in specific settings. We refer to the Accelerated Inexact DanE as "Aide" (Algorithm 28). The key idea is to apply the generic acceleration scheme—catalyst—in [88] to InexactDane. We show that by a careful selection of τ in Algorithm 28, one can achieve optimal communication complexity in interesting and important settings.

11.5.1 Quadratic case

Here we show that by appropriate selection of τ, for quadratic case, one can achieve faster convergence rates that match the lower bounds presented in [175].

Theorem 11.5.1. *Assume that $\lambda I \preceq H \preceq LI$ and further assume that $\{F_k\}$ are δ-related. Then the iterates of* Aide *(Algorithm 28) with $\eta = 1$, $\mu = 0$, $\tau = \max\{0, 2\sqrt{2}\delta - \lambda\}$, $s = \tilde{O}(1)$ and $\gamma = 1/8 - (\delta^2/2(\tau+\lambda)^2)$ after $t = \tilde{O}(\sqrt{\delta/\lambda}\log(1/\epsilon))$ satisfy $f(x^t) \leq f(\hat{x}) + \epsilon$ and the total number of iterations in* InexactDane *(Option I) is $\tilde{O}(\sqrt{\delta/\lambda}\log(1/\epsilon))$.*

```
Algorithm 28: AIDE ($f$, $x^0$, $\lambda$, $\tau$, $s$, $\gamma$, $\mu$, $\epsilon$)
Data: $f(x) = \frac{1}{K}\sum_{k=1}^{K} F_k(x)$, Initial point $y^0 = x^0 \in \mathbb{R}^d$, INEXACTDANE iterations $s$,
      inexactness parameter $0 \le \gamma < 1, \tau \ge 0$
Let $q = \lambda/(\lambda+\tau)$
while $f(x^{t-1}) - f(\hat{x}) \le \epsilon$ do
    Define $f^t(x) := \frac{1}{K}\sum_{k=1}^{K}(F_k(x) + \frac{\tau}{2}\|x - y^{t-1}\|^2)$
    $x^t = \text{INEXACTDANE}(f^t, x^{t-1}, s, \gamma, \mu)$
    Find $\zeta_t \in (0,1)$ such that $\zeta_t^2 = (1-\zeta_t)\zeta_{t-1}^2 + q\zeta_t$
    Compute $y^t = x^t + \beta_t(x^t - x^{t-1})$ where $\beta_t = \frac{\zeta_{t-1}(1-\zeta_{t-1})}{\zeta_{t-1}^2+\zeta_t}$
end
return $x^t$
```

Note that the communication cost of AIDE is proportional to the number of iterations of INEXACTDANE executed as part of AIDE algorithm. Hence, the total communication complexity of AIDE is $\tilde{O}(\sqrt{\delta/\lambda}\log(1/\epsilon))$. It is interesting to note that it is enough to solve each subproblem in INEXACTDANE up to a constant accuracy of $\gamma = 1/8$ in order to achieve the convergence rate stated above. In other words, by using AIDE, one need not solve the subproblems with high accuracy and still achieve faster convergence than INEXACTDANE.

11.5.2 Convex case

For the general strongly convex case, we prove the following key result concerning the number of iterations of AIDE and INEXACTDANE used as part of AIDE.

Theorem 11.5.2. *Assume that for all $k \in [K]$, the function F_k is L-smooth and λ-strongly convex. Then the iterates of* AIDE *with parameters λ, $\eta = 1$, $\mu = 12L$, $\gamma = 1/8$, $s = \tilde{O}(1)$ and $\tau = L - \lambda$ after $t = \tilde{O}(\sqrt{L/\lambda}\log(1/\epsilon))$ iterations satisfy $f(x^t) \le f(\hat{x}) + \epsilon$ and the total number of iterations in* INEXACTDANE *(Option II) is $\tilde{O}(\sqrt{L/\lambda}\log(1/\epsilon))$.*

The upper bound on the communication complexity matches the lower bounds proved in [175] for unrelated strongly convex functions. It is an interesting open problem to obtain better communication complexity for a subclass of strongly convex functions other than the quadratic case explored above.

As in the case of INEXACTDANE, we apply AIDE to a perturbed problem when f is weakly convex. In this manner, we invoke catalyst (acceleration) for strongly convex functions. Recall the function f_ϵ is defined as $f_\epsilon(x) = f(x) + \frac{\epsilon}{2}\|x - x^0\|^2$. Using AIDE on f_ϵ, Theorem 11.5.2 shows that one can achieve ϵ-accuracy on function f_ϵ in $t = \tilde{O}(\sqrt{(L+\epsilon)/\epsilon}\log(1/\epsilon))$ iterations. This follows from the fact that f_ϵ is $L+\epsilon$-smooth and ϵ-strongly convex. Again, using similar argument as in Equation (11.17), we have the following corollary.

Corollary 11.5.2.1. *Suppose the function f is weakly convex. Then iterates of* AIDE *in Algorithm 28 applied on f_ϵ with $\lambda = \epsilon$, $\eta = 1$, $\gamma = 1/8$ and $\mu = 12(L+\epsilon)$, $s = \tilde{O}(1)$ and $\tau = L$*

after $t = \tilde{O}(\sqrt{L/\epsilon}\log(1/\epsilon))$ *iterations satisfy* $f(x^t) \leq f(\hat{x}) + O(\epsilon)$.

11.6 Connection to a Practical Distributed Version of SVRG

In this section, we discuss the connection between SGD, INEXACTDANE and a version of distributed SVRG that was observed to perform well in practice [76], but has not yet been successfully analyzed. Algorithm 27 uses an approximate solution to the minimization of function $g^t_{k,\mu}(x)$ at each iteration; however, it does not specify the algorithm (local solver) used to obtain it. While one could use any of recent fast algorithms for finite sums [32, 33, 153, 156] with comparable results, we highlight the consequences of applying the SVRG algorithm [71, 74] as a local solver in INEXACTDANE.

The SVRG algorithm for minimizing average of functions $f(x) = \frac{1}{N}\sum_{i=1}^{N} f_i(x)$ runs in two nested loops. In the t^{th} outer iteration, SVRG computes the gradient of the entire function $\nabla f(x^{t-1})$ followed by inner loop, where an update direction at x is computed as $\nabla f_i(x) - \nabla f_i(x^{t-1}) + \nabla f(x^{t-1})$ for a randomly chosen index $i \in [N]$. For its distributed version, the inner loop is executed parallelly over all the machines but the index i for machine k is sampled only from $\mathcal{P}_k$ (the data available locally) instead of $[N]$. For completeness, the pseudocode is provided in Appendix 11.12.

We can obtain an identical sequence[1] of iterates x^t by looking at a variation of INEXACTDANE algorithm. It is easy to verify that if we apply SVRG to the INEXACTDANE subproblem $g^t_{k,\mu}$ in (11.2) with $\mu = 0$ and $\eta = 1$, running SVRG for a single outer iteration, the resulting procedure is equivalent to running the Algorithm 29 (in the Appendix). Pointing out this connection gives partial justification for the performance of Algorithm 29. However, a direct analysis for Algorithm 29 still remains an open question as the above reasoning applies only when $\mu = 0$ and $\eta = 1$. Note that our results apply to this special setting only under specific conditions; for example when quadratic functions $\{F_k\}$ that are δ-related with sufficiently small δ.

11.7 Experiments

In this section, we demonstrate the empirical performance of the INEXACTDANE and AIDE algorithms on binary classification task using different loss functions. We compare the performance of our algorithms to that of COCOA+, a popular distributed algorithm [95]. COCOA+ is particularly relevant to our setting since, similar to INEXACTDANE and AIDE, it provides flexibility in dealing with communication/computation tradeoff by choosing the local computation effort spent between rounds of communication. We use SVRG as the local solver for INEXACTDANE and AIDE, and SDCA as the local solver for COCOA+ in all our experiments. Unless stated otherwise, at each iteration of the algorithms, we run the corresponding local solvers for a single pass over data available

[1]Subject to the same sequence on sampled indices $i \in \mathcal{P}_k$.

locally. Note that for all the algorithms considered here, the iteration complexity is proportional to their communication complexity.

For our experiments, we use standard binary classification datasets rcv1, covtype, real-sim and url[2]. As part of preprocessing, we normalize the data and add a bias term. In our experiments, the data is randomly partitioned across individual nodes; thus, mimicking the i.i.d data distribution. We are interested in examining the effect of varying amount of local computation and number of nodes on the performance of the algorithms. For our results, we present the best performance obtained by selected from a range of stepsize parameters for SVRG and aggregation parameters for COCOA+. For COCOA+, this amounts to searching for the optimal choice of parameter σ'. In the plots, we use DANE as the label for INEXACTDANE.

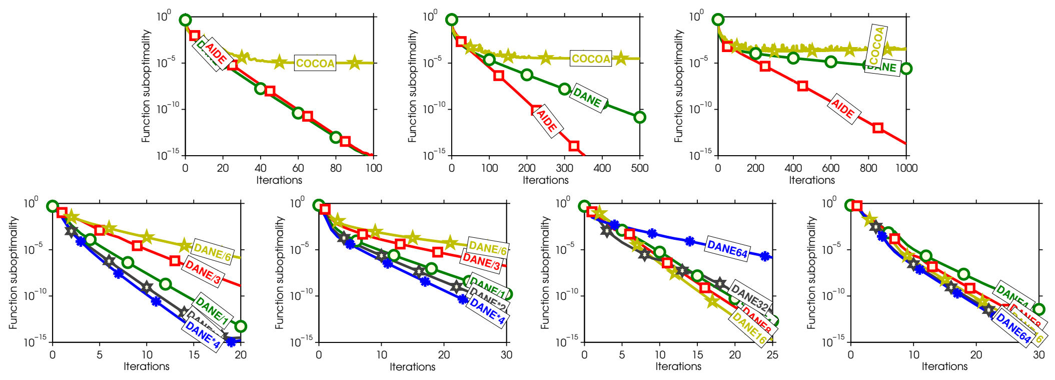


Figure 11.1: Top: rcv1 dataset; Smoothed hinge loss; λ set to $1/(cN)$, for $c \in \{1, 10, 100\}$. Bottom: Logistic loss; *left-hand two:* Varying number of local data passes per iteration. rcv1 and url datasets. *right-hand two:* Varying number of nodes; rcv1 and url datasets.

In the top row of Figure 11.1, we compare INEXACTDANE, AIDE and COCOA+ for classification task on the rcv1 dataset with smoothed hinge loss on 8 nodes. The three plots are with regularization parameter λ set to $1/(cN)$, for $c \in \{1, 10, 100\}$. It can be seen that AIDE outperforms both INEXACTDANE and COCOA+. We see similar behavior on other datasets (see Section 11.13.1 of the Appendix). The benefits of AIDE are particularly pronounced in settings with large condition number $\kappa = L/\lambda > N$. This can also be explained theoretically, as convergence rate of fast accelerated stochastic methods has a dependence of $O(\sqrt{N\kappa})$ on κ and N, as opposed to $O(\kappa)$ without acceleration [88, 154].

In the two left-hand side plots of the bottom row of Figure 11.1, we demonstrate the effect of local computation effort at each iteration, to solve the local subproblem (11.2), on the overall convergence of the algorithm. The different lines signify $\{1/6, 1/3, 1, 2, 4\}$ passes through data available locally per iteration of INEXACTDANE. From the plots, it can be seen that running SVRG beyond 4 passes through local data only provides little improvement; thus, suggesting a natural computational setting for INEXACTDANE and demonstrating its advantage over DANE. Finally, in the two right-hand side plots of the

[2]Datasets available at https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/

bottom row, we show performance of InexactDane with increasing number of nodes. In particular, we partition the data to $\{4, 8, 16, 32, 64\}$ nodes, and keep the number of local iterations of Svrg constant for all settings. The results suggest that in practice, InexactDane can scale gracefully with the number of nodes. We only observe drop in the performance with rcv1 dataset on 64 nodes, where the optimal stepsize of Svrg was significantly smaller than in other cases. We relegate experiments on other datasets and loss functions, and demonstration of the robustness of the InexactDane method over Dane to Section 11.13 of the Appendix.

11.8 Discussion

In this section, we give a brief comparison of the results we presented. In particular, we compare the key aspects of Dane, Disco, Cocoa+, InexactDane and Aide .

Communication Complexity: We showed that Aide nearly achieves the lower bounds proved in [175] in specific cases. When the functions are quadratic δ-related, Aide and Disco match the communication complexity lower bound of $\tilde{O}(\sqrt{\delta/\lambda}\log(1/\epsilon))$. However, in this case, Dane and InexactDane have a sub-optimal communication complexity of $\tilde{O}((\delta^2/\lambda^2)\log(1/\epsilon))$. Similarly, for the general unrelated strongly convex function with condition number κ, Aide and Disco enjoy communication complexity of $\tilde{O}(\sqrt{\kappa}\log(1/\epsilon))$ [175]. Again, this complexity is superior to the convergence rate of $O(\kappa\log(1/\epsilon))$ of Cocoa+ , InexactDane and Dane.

Nature of oracle access: Both Dane and Disco require access to a strong oracle. While Dane technically requires an oracle that solves the subproblem (11.2), Disco requires a second-order oracle for its execution. On the other hand, both InexactDane and Aide can be executed using a simple first-order oracle, matching the major advantage of Cocoa+.

Parallelism and Implementation: One of the appealing aspect of the Dane, Inexact-Dane, Aide and Cocoa+ is the simplicity of implementation and its suitability for large-scale distributed environments. Each iteration of these algorithms is embarrassingly parallelizable since it involves solving a local objective function at each node. The same cannot be said about Disco due to the asymmetric workload on the master node at each iteration of the inexact damped Newton iteration.

Distributed SVRG: As a by-product of our analysis, we obtain partial convergence guarantees of a distributed version of popular Svrg algorithm, that was observed to perform well in practice [76].

In conclusion, Aide adopts practical advantages of Dane, but at the same time also achieves the optimal communication complexity in [175]. Furthermore, similar to Cocoa+, Aide and InexactDane provide an efficient way of balancing communication and computation complexity; thereby, providing a powerful framework for solving large-scale distributed optimization in a communication-efficient manner.

Appendix: Omitted Proofs and Additional Experiments

In this section, we present the omitted proofs of InexactDane and Aide algorithms, and provide additional experiments for this chapter.

11.9 Analysis of InexactDane

Quadratic case

Before proving the main result, we will need to establish two lemmas.

Lemma 11.9.1. *Let assumptions of Theorem 11.3.1 be satisfied. Then*

$$\left\| x^t - \frac{1}{K}\sum_{k=1}^{K} \hat{x}_k^t \right\| \leq \frac{\eta\gamma}{K}\sum_{k=1}^{K} \|(H_k+\mu I)^{-1}H\| \|x^{t-1}-\hat{x}\|$$

Proof. Since $\hat{x}_k^t = \arg\min_x g_{k,\mu}^t(x)$, we have $\nabla g_{k,\mu}^t(\hat{x}_k^t) = 0$. That is, $0 = \nabla g_{k,\mu}^t(\hat{x}_k^t) = \nabla F_k(\hat{x}_k^t) + \eta \nabla f(x^{t-1}) - \nabla F_k(x^{t-1}) + \mu(\hat{x}_k^t - x^{t-1})$. By rearranging the terms in the last expression, we get

$$\hat{x}_k^t = x^{t-1} - \eta\left((H_k+\mu I)^{-1} H x^{t-1} + (H_k+\mu I)^{-1} l \right). \tag{11.5}$$

Taking norms on both sides of (11.5), and using the fact that $\hat{x} = -H^{-1}l$, we obtain

$$\|x^{t-1} - \hat{x}_k^t\| = \eta \|(H_k+\mu I)^{-1}H(x^{t-1}-\hat{x})\|. \tag{11.6}$$

We proceed to prove the desired result by using the following set of inequalities:

$$\begin{aligned}
\left\| x^t - \frac{1}{K}\sum_{k=1}^{K} \hat{x}_k^t \right\| &\leq \frac{1}{K}\sum_{k=1}^{K} \|x_k^t - \hat{x}_k^t\| \\
&\leq \frac{1}{K}\sum_{k=1}^{K} \gamma \left\| x^{t-1} - \hat{x}_k^t \right\| \\
&= \frac{\eta\gamma}{K}\sum_{k=1}^{K} \|(H_k+\mu I)^{-1}H(x^{t-1}-\hat{x})\| \\
&\leq \frac{\eta\gamma}{K}\sum_{k=1}^{K} \|(H_k+\mu I)^{-1}H\| \|x^{t-1}-\hat{x}\|.
\end{aligned} \tag{11.7}$$

The first inequality follows from Jensen's inequality and the fact that $x^t = \sum_{k=1}^{K} x_k^t / K$. The second inequality follows from the approximation condition on x_k^t. The equality is obtained from Equation (11.6). The last inequality follows from properties of spectral norm of a matrix. □

Lemma 11.9.2. *Let assumptions of Theorem 11.3.1 be satisfied. Then*

$$\|x^t - \hat{x}\| - \left\|\eta \tilde{H}^{-1} H - I\right\| \|x^{t-1} - \hat{x}\| \leq \left\| x^t - \frac{1}{K}\sum_{k=1}^{K} \hat{x}_k^t \right\|.$$

Proof.

$$\begin{aligned}
\left\| x^t - \frac{1}{K}\sum_{k=1}^{K} \hat{x}_k^t \right\| &= \left\| x^t - x^{t-1} + \frac{\eta}{K}\sum_{k=1}^{K} (H_k + \mu I)^{-1} H (x^{t-1} - \hat{x}) \right\| \\
&= \left\| (x^t - \hat{x}) - (x^{t-1} - \hat{x}) + \eta \tilde{H}^{-1} H (x^{t-1} - \hat{x}) \right\| \\
&= \left\| (x^t - \hat{x}) - (I - \eta \tilde{H}^{-1} H)(x^{t-1} - \hat{x}) \right\| \\
&\geq \|x^t - \hat{x}\| - \left\| (I - \eta \tilde{H}^{-1} H)(x^{t-1} - \hat{x}) \right\| \\
&\geq \|x^t - \hat{x}\| - \left\| I - \eta \tilde{H}^{-1} H \right\| \|x^{t-1} - \hat{x}\|. \qquad (11.8)
\end{aligned}$$

The first equality follows from the optimality property of $\hat{x}_k^t$ (see Equation (11.5)). The first inequality follows from the triangle inequality. The second inequality follows from the properties of the matrix spectral norm. □

11.9.1 Proof of Theorem 11.3.1

Proof. Lemmas 11.9.1 and 11.9.2 imply the inequality

$$\left\|x^t - \hat{x}\right\| - \left\|\eta \tilde{H}^{-1} H - I\right\| \left\|x^{t-1} - \hat{x}\right\| \leq \frac{\eta\gamma}{K}\sum_{k=1}^{K} \left\|(H_k + \mu I)^{-1} H\right\| \left\|x^{t-1} - \hat{x}\right\|.$$

It suffices to rearrange the terms. □

11.9.2 Proof of Theorem 11.3.3

Proof. Follows by using Lemma 11.11.2 (see Appendix 11.11) together with Theorem 11.3.1. □

11.9.3 InexactDane for Stochastic Quadratic Setting

In the interesting case of stochastic quadratic setting (see [163]), we can provide a more precise result. More specifically, we have the following key result in the stochastic quadratic setting.

Corollary 11.9.2.1. *Suppose each H_k is the average of n i.i.d. symmetric positive definite matrices with largest eigenvalue bounded above by L. Then for any $0 < \alpha \leq 1$, with probability $1 - \alpha$ we have:*

i Functions $\{F_k\}$ are δ-related with $\delta = \sqrt{\frac{32L^2 \log(Kd/\alpha)}{n}}$,

ii The iterates of INEXACTDANE *(Algorithm 27) (with parameters defined in Theorem 11.3.3) after*

$$t = \tilde{O}\left(\frac{(L/\lambda)^2}{n} \log\left(\frac{Kd}{\alpha}\right) \log\left(\frac{L\|x^0 - \hat{x}\|^2}{\epsilon}\right)\right)$$

satisfy $f(x^t) \leq f(\hat{x}) + \epsilon$.

Proof. The first part of the proof is directly from Lemma 2 of [163]. The second part of the proof follows from Theorem 11.3.3. □

Strongly Convex Case

11.9.4 Proof of Theorem 11.4.1

Proof. We first observe that

$$\begin{aligned} \|\nabla g^t_{k,\mu}(x^t_k) - \nabla g^t_{k,\mu}(x^{t-1})\| &\geq \|\nabla g^t_{k,\mu}(x^{t-1})\| - \|\nabla g^t_{k,\mu}(x^t_k)\| \\ &\geq (1-\gamma)\|\nabla g^t_{k,\mu}(x^{t-1})\|. \end{aligned} \tag{11.9}$$

The first inequality follows from triangle inequality. The second inequality follows from the condition that $\|\nabla g^t_{k,\mu}(x^t_k)\| \leq \gamma \|\nabla g^t_{k,\mu}(x^{t-1})\|$. We now define function $h^t_k : \mathbb{R}^d \to \mathbb{R}$ as

$$h^t_k(x) := F_k(x) + \frac{\mu}{2}\|x - x^{t-1}\|^2.$$

We define the virtual iterate $\hat{x}^t_k = \arg\min_x g^t_{k,\mu}(x)$. Using the optimality conditions of $\hat{x}^t_k$, we have the following:

$$\nabla h^t_k(\hat{x}^t_k) - \nabla h^t_k(x^{t-1}) = -\eta \nabla f(x^{t-1}). \tag{11.10}$$

Also, define Bregman divergence of a smooth function ϕ as $D_\phi(x, x') = \phi(x) - \phi(x') - \langle \nabla\phi(x'), x - x'\rangle$. We observe the following:

$$\begin{aligned} f(x^t_k) &= f(x^{t-1}) + \langle \nabla f(x^{t-1}), x^t_k - x^{t-1}\rangle + D_f(x^t_k, x^{t-1}) \\ &= f(x^{t-1}) - \frac{1}{\eta}\langle \nabla h^t_k(\hat{x}^t_k) - \nabla h^t_k(x^{t-1}), x^t_k - x^{t-1}\rangle + D_f(x^t_k, x^{t-1}) \\ &= f(x^{t-1}) - \frac{1}{\eta}\underbrace{\langle \nabla h^t_k(\hat{x}^t_k) - \nabla h^t_k(x^t_k), x^t_k - x^{t-1}\rangle}_{T_1} \\ &\qquad - \frac{1}{\eta}\langle \nabla h^t_k(x^t_k) - \nabla h^t_k(x^{t-1}), x^t_k - x^{t-1}\rangle + D_f(x^t_k, x^{t-1}). \end{aligned} \tag{11.11}$$

The second equality is due to optimality condition in (11.10). We bound the term T_1 in the following manner:

$$\begin{aligned} |T_1| &\leq \|\nabla h_k^t(\hat{x}_k^t) - \nabla h_k^t(x_k^t)\| \|x_k^t - x^{t-1}\| \\ &\leq (L+\mu)\|\hat{x}_k^t - x_k^t\|\|x_k^t - x^{t-1}\|. \end{aligned} \tag{11.12}$$

The first inequality is due to Cauchy-Schwarz inequality. The second inequality follows from $L+\mu$ lipschitz continuous nature of the gradient of h_k^t. In order to proceed further, a bound on the term $\|x_k^t - x^{t-1}\|$ is obtained in the following fashion:

$$\begin{aligned} &\|x^{t-1} - x_k^t\| \leq \|x^{t-1} - \hat{x}_k^t\| + \|\hat{x}_k^t - x_k^t\| \\ &\leq \|x^{t-1} - \hat{x}_k^t\| + \gamma\|\hat{x}_k^t - x^{t-1}\| = (1+\gamma)\|x^{t-1} - \hat{x}_k^t\|. \end{aligned} \tag{11.13}$$

The first inequality is due to triangle inequality. The second inequality is due to the inexactness condition in Algorithm 27. Substituting the above bound in Equation (11.12), we have the following:

$$|T_1| \leq (L+\mu)(1+\gamma)\|\hat{x}_k^t - x_k^t\|\|x^{t-1} - \hat{x}_k^t\| \leq (L+\mu)(1+\gamma)\gamma\|x^{t-1} - \hat{x}_k^t\|^2. \tag{11.14}$$

The second inequality is again due to inexactness condition in Algorithm 27. In order to bound $\|x^{t-1} - \hat{x}_k^t\|^2$, we observe the following:

$$(\mu+\lambda)\|x^{t-1} - \hat{x}_k^t\| \leq \|\nabla g_{k,\mu}^t(x^{t-1})\| = \|\eta\nabla f(x^{t-1})\|. \tag{11.15}$$

The first inequality follows from Lemma 11.11.3. Using this relationship in Equation (11.14), we have the following:

$$|T_1| \leq \frac{2\gamma\eta^2(L+\mu)}{(\lambda+\mu)^2}\|\nabla f(x^{t-1})\|^2.$$

Substituting the bound in Equation (11.11), we have:

$$\begin{aligned}
f(x_k^t) &\leq f(x^{t-1}) + \frac{2\gamma\eta(L+\mu)}{(\lambda+\mu)^2}\|\nabla f(x^{t-1})\|^2 \\
&\quad - \frac{1}{\eta(L+\mu)}\|\nabla h_k^t(x_k^t) - \nabla h_k^t(x^{t-1})\|^2 + D_f(x_k^t, x^{t-1}) \\
&\leq f(x^{t-1}) + \frac{2\gamma\eta(L+\mu)}{(\lambda+\mu)^2}\|\nabla f(x^{t-1})\|^2 \\
&\quad - \frac{1}{\eta(L+\mu)}\|\nabla h_k^t(x_k^t) - \nabla h_k^t(x^{t-1})\|^2 + \frac{L}{2}\|x^{t-1} - x_k^t\|^2 \\
&\leq f(x^{t-1}) + \frac{2\gamma\eta(L+\mu)}{(\lambda+\mu)^2}\|\nabla f(x^{t-1})\|^2 \\
&\quad - \frac{1}{\eta(L+\mu)}\|\nabla g_{k,\mu}^t(x_k^t) - \nabla g_{k,\mu}^t(x^{t-1})\|^2 + \frac{2L\eta^2}{(\lambda+\mu)^2}\|\nabla f(x^{t-1})\|^2 \\
&\leq f(x^{t-1}) - \left[\frac{(1-\gamma)^2}{\eta(L+\mu)} - \frac{2L}{(\lambda+\mu)^2} - \frac{2\gamma(L+\mu)}{\eta(\lambda+\mu)^2}\right]\eta^2\|\nabla f(x^{t-1})\|^2 \\
&\leq f(x^{t-1}) - \left[\frac{(1-\gamma)^2}{\eta(L+\mu)} - \frac{2L}{(\lambda+\mu)^2} - \frac{2\gamma(L+\mu)}{\eta(\lambda+\mu)^2}\right]\eta^2\lambda(f(x^{t-1}) - f(\hat{x})). \qquad (11.16)
\end{aligned}$$

The first step is due to Lemma 11.11.3. The second step follows the L-smoothness of f. The third step follows from the fact that $\|\nabla g_{k,\mu}^t(x_k^t) - \nabla g_{k,\mu}^t(x^{t-1})\| = \|\nabla h_k^t(x_k^t) - \nabla h_k^t(x^{t-1})\|$ and bound on $\|x^{t-1} - x_k^t\|^2$ from Equation (11.13) and (11.15). The fourth inequality follows from Equation (11.9) and the fact that $\nabla g_{k,\mu}^t(x^{t-1}) = \nabla f(x^{t-1})$. The last inequality is due to strong convexity of function f. Finally, we observe:

$$(f(x^t) - f(\hat{x}) \leq \frac{1}{K}\sum_{k=1}^{K}(f(x_k^t) - f(\hat{x})) \leq (1-\tilde{\rho})(f(x^{t-1}) - f(\hat{x})).$$

The first inequality is due to convexity. The second inequality follows from Equation (11.16). Therefore, we have the required result. □

Weakly Convex Case

Proof of Corollary 11.4.1.1

We observe the following:

$$\begin{aligned}
f(x^t) \leq f_\epsilon(x^t) \leq \min_x f_\epsilon(x) + (1-\rho_\epsilon)^s[f_\epsilon(x^0) - \min_x f_\epsilon(x)] \\
\leq f(\hat{x}) + \frac{\epsilon}{2}\|\hat{x} - x^0\|^2 + (1-\rho_\epsilon)^s[f_\epsilon(x^0) - \min_x f_\epsilon(x)] \\
\leq f(\hat{x}) + \epsilon\left[(1/2)\|\hat{x} - x^0\|^2 + (f(x^0) - f(\hat{x}))\right]. \qquad (11.17)
\end{aligned}$$

The second inequality is a consequence of Equation (11.4). The last inequality follows from the facts that (i) $s = \tilde{O}(L \log(1/\epsilon)/\epsilon)$ and (ii) $[f_\epsilon(x^0) - \min_x f_\epsilon(x)] = [f(x^0) - \min_x f_\epsilon(x)] \leq [f(x^0) - f(\hat{x})]$.

Nonconvex Case

11.9.5 Proof of Theorem 11.4.2

Proof. Using essentially the same argument as in Theorem 11.4.1, we have the following:

$$f(x_k^t) \leq f(x^{t-1}) - \left[\frac{(1-\gamma)^2}{\eta(L+\mu)} - \frac{2L}{(\mu-L)^2} - \frac{2\gamma(L+\mu)}{\eta(\mu-L)^2}\right] \eta^2 \|\nabla f(x^{t-1})\|^2.$$

The argument uses the fact that $g_{k,\mu}^t$ is $(\mu + L)$-smooth and $(\mu - L)$-strongly convex. This is in turn due to the lipschitz continuous nature of the gradient. Note that this is possible only when $\mu > L$ (as mentioned in the theorem statement). Using the definition of x^t in the nonconvex case, we have

$$f(x^t) \leq f(x^{t-1}) - \left[\frac{(1-\gamma)^2}{\eta(L+\mu)} - \frac{2L}{(\mu-L)^2} - \frac{2\gamma(L+\mu)}{\eta(\mu-L)^2}\right] \eta^2 \|\nabla f(x^{t-1})\|^2. \tag{11.18}$$

Rearranging the above inequality and using the definition of θ, we get

$$\|\nabla f(x^{t-1})\|^2 \leq \frac{f(x^{t-1}) - f(x^t)}{\theta}.$$

Using a telescopic sum on the above inequality and the fact that $\hat{x}$ is an optimal solution of (11.1), we have the desired result. □

11.10 Analysis of AIDE

Quadratic Case

11.10.1 Proof of Theorem 11.5.1

Proof. First consider the case where $2\sqrt{2}\delta \leq \lambda$. In this case, we have $\tau = 0$ and $1/16 \leq \gamma \leq 1/8$. The required result trivially follows from Theorem 11.3.3. We turn our attention to the interesting case of $2\sqrt{2}\delta > \lambda$. For this case, we choose $\tau = 2\sqrt{2}\delta - \lambda$. We use $F_k^t(x)$ to denote the function $F_k(x) + (\tau/2)\|x - y^{t-1}\|^2$. As mentioned in pseudocode of Algorithm 28, we use INEXACTDANE for the set of functions F_k^t. Let $\hat{u}^t = \arg\min_x f^t(x)$ (refer to Algorithm 28). By solving each subproblem inexactly for s iterations, from Theorem 11.3.3, we have the following:

$$f^t(x^t) - f^t(\hat{u}^t) \leq \tfrac{L+\tau}{2}\|x^t - \hat{u}^t\|^2 \leq \tfrac{L+\tau}{2}\tilde{\rho}^s \|x^{t-1} - \hat{u}^t\|^2, \tag{11.19}$$

where $\tilde{\rho} = 2/3$. The first inequality follows from the $L + \tau$ Lipschitz continuous gradient of f^t. The second inequality follows from Theorem 11.3.3. The value of $\tilde{\rho}$ is obtained from Theorem 11.3.3 noting the following two facts. (i) The inexactness parameter $\gamma \leq 1/8$. (ii) $\tilde{\rho} = 2/3$ when $\gamma \leq 1/8$ and $2\sqrt{2}\delta \leq (\lambda + \tau)$. Here, we used the fact that INEXACTDANE is applied on the function f^t and $\nabla^2 f^t \succeq (\lambda + \tau)I$. Using Equation (11.19), we have the following:

$$f^t(x^t) - f^t(\hat{u}^t) \leq \tfrac{L+\tau}{2}\tilde{\rho}^s \|x^{t-1} - \hat{u}^t\|^2 \leq \tfrac{L+\tau}{\lambda+\tau}\tilde{\rho}^s [f^t(x^{t-1}) - f^t(\hat{u}^t)].$$

This simply follows from the fact that $\nabla^2 f^t \succeq (\lambda + \tau)I$. Using Proposition 3.2 of [88], we have the total number of iterations of INEXACTDANE iterations one has to execute to achieve ϵ-accuracy is

$$t = \tilde{O}\left(\tfrac{1}{1-\tilde{\rho}}\sqrt{\tfrac{\lambda+\tau}{\lambda}}\log(1/\epsilon)\right) = \tilde{O}\left(\sqrt{\tfrac{\delta}{\lambda}}\log(1/\epsilon)\right).$$

This is obtained from the fact that $\tilde{\rho} = 2/3$. Therefore, we get the desired result. □

11.10.2 AIDE in the Stochastic Quadratic Setting

The following result follows as a Corollary of Theorem 11.5.1.

Corollary 11.10.0.1. *Suppose each H_k is the average of n i.i.d positive semidefinite matrices with eigenvalues at most L. Then with probability at least $1 - \alpha$:*

i Functions $\{F_k\}$ are δ-related with $\delta = \sqrt{\frac{32L^2 \log(Kd/\alpha)}{n}}$,

ii The iterates of INEXACTDANE *(Algorithm 27) (with parameters in Theorem 11.5.1) after*

$$t = \tilde{O}\left(\tfrac{\sqrt{L}}{n^{1/4}\sqrt{\lambda}}\log^{1/4}\left(\tfrac{Kd}{\alpha}\right)\log\left(\tfrac{L\|x^0 - \hat{x}\|^2}{\epsilon}\right)\right)$$

satisfy $f(x^t) \leq f(\hat{x}) + \epsilon$.

Proof. The first part of the proof is directly from Lemma 2 of [163]. The second part of the proof follows from Theorem 11.5.1. □

Strongly convex case

11.10.3 Proof of Theorem 11.5.2

Proof. Let $\hat{u}^t = \arg\min_x f^t(x)$. We first observe that after s iterations of INEXACTDANE in the t^{th} iteration of AIDE (applied on f^t), we have the following:

$$f^t(x^t) - f^t(\hat{u}^t) \leq (1 - \tilde{\rho})^s (f^t(x^{t-1}) - f^t(\hat{u}^t)), \tag{11.20}$$

where

$$\tilde{\rho} = \left[\frac{49}{64(L+\mu+\tau)} - \frac{2(L+\tau)}{(\lambda+\tau+\mu)^2} - \frac{(L+\tau+\mu)}{4(\lambda+\tau+\mu)^2}\right] L$$

This follows directly from Theorem 11.4.1 and the fact that f^t has $(L+\tau)$-Lipschitz continuous gradient and is $(\lambda+\tau)$- strongly convex. Note that with the given value of τ, μ and γ, we have following:

$$\tilde{\rho} = L\left[\frac{49}{64(2L+\mu-\lambda)} - \frac{2(2L-\lambda)}{(L+\mu)^2} - \frac{2L+\mu-\lambda}{4(L+\mu)^2}\right] \tag{11.21}$$

$$\geq L\left[\frac{49}{64(14L)} - \frac{4L}{(13L)^2} - \frac{14L}{4(13L)^2}\right] \geq 0.01. \tag{11.22}$$

The above bound follows from simple algebra. Now again using Proposition 3.2 of [88], the total number of InexactDane iterations required to achieve an ϵ-accurate solution is

$$t = \tilde{O}\left(\frac{1}{\tilde{\rho}}\sqrt{\frac{\lambda+\tau}{\lambda}}\log\left(\frac{1}{\epsilon}\right)\right) = \tilde{O}\left(\sqrt{\frac{L}{\lambda}}\log\left(\frac{1}{\epsilon}\right)\right).$$

This is obtained from the bound in Equation (11.21). This gives us the desired result. Note that the constants in this result are not optimized. □

11.11 Auxiliary Results

Here we establish two lemmas which are used in the proofs of the main results.

The following result is a slight extension of Lemma 4 in [163].

Lemma 11.11.1. *Let μ, δ, ξ be positive constants satisfying $\max\{\mu, \delta\} < \xi$. Further, let A and $B_1, \ldots, B_K$ be $d \times d$ symmetric matrices satisfying $\xi I \preceq A$, $\sum_k B_k = 0$ and $\|B_k\| \leq \delta$ for all $k \in [K]$. Then, we have the following:*

$$\left\|\left((A+B_k)^{-1} - A^{-1}\right)(A-\mu I)\right\| \leq \frac{2\delta}{\xi-\delta}, \qquad k \in [K],$$

$$\left\|\left(\frac{1}{K}\sum_{k=1}^{K}(A+B_k)^{-1} - A^{-1}\right)(A-\mu I)\right\| \leq \frac{2\delta^2}{\xi(\xi-\delta)}.$$

Proof. First, we observe that

$$(A+B_k)^{-1} = (A(I+A^{-1}B_k))^{-1} = (I+A^{-1}B_k)^{-1}A^{-1} = A^{-1} + \sum_{j=1}^{\infty}(-1)^j(A^{-1}B_k)^jA^{-1}.$$

The above equality follows form series expansion of $(I+A^{-1}B_k)^{-1}$, which is possible as $\|A^{-1}B_k\| \leq \|A^{-1}\|\|B_k\| = (\delta/\xi) < 1$. From the above equality, we obtain the following bound:

$$\left\| \left((A+B_k)^{-1} - A^{-1} \right) \left(A - \mu I \right) \right\| = \left\| \sum_{j=1}^{\infty} (-1)^j (A^{-1}B_k)^j A^{-1}(A-\mu I) \right\|$$
$$\leq \sum_{j=1}^{\infty} \left\| (-1)^j (A^{-1}B_k)^j A^{-1}(A-\mu I) \right\|$$
$$\leq \sum_{j=1}^{\infty} \left\| A^{-1} \right\|^j \|B_k\|^j \left\| I - \mu A^{-1} \right\|$$
$$\leq \sum_{j=1}^{\infty} \frac{\delta^j}{\xi^j} \left(1 + \frac{\mu}{\xi} \right) \leq \frac{2\delta}{\xi - \delta}.$$

The first inequality follows from the triangle inequality. The second inequality follows from the Cauchy-Schwarz inequality. The last inequality follows from the fact that $\mu/\xi < 1$.

The second part of the claim was established as Lemma 4 in [163]. □

Lemma 11.11.2. *Assume that $\lambda I \preceq H \preceq LI$, where $H := \frac{1}{K}\sum_{k=1}^{K} H_k$ and $\lambda > 0$. Further, assume that $\|H_k - H\| \leq \delta$ for all $k \in [K]$ and let $\tilde{H}^{-1} := \frac{1}{K}\sum_{k=1}^{K}(H_k + \mu I)^{-1}$, where $\mu = \max\left\{0, 8\frac{\delta^2}{\lambda} - \lambda\right\}$. If we let*

$$\gamma := \begin{cases} \frac{1}{8} & \text{if} \quad 2\sqrt{2}\delta \leq \lambda, \\ \frac{\lambda^2}{192\delta^2} & \text{otherwise,} \end{cases}$$

then

$$\rho = \|\tilde{H}^{-1}H - I\| + \frac{\gamma}{K}\sum_{k=1}^{K} \|(H_k + \mu I)^{-1}H\| \leq \begin{cases} \frac{2}{3} & \text{if} \quad 2\sqrt{2}\delta \leq \lambda, \\ 1 - \frac{\lambda^2}{24\delta^2} & \text{otherwise.} \end{cases}$$

Proof. Using Lemma 11.11.1 with $A = H + \mu I$ and $B_k = H_k - H$, we have the following inequality:

$$\left\| \left((H_k + \mu I)^{-1} - (H + \mu I)^{-1} \right) H \right\| \leq \frac{2\delta}{\lambda + \mu - \delta}, \tag{11.23}$$

$$\left\| \left(\tilde{H}^{-1} - (H + \mu I)^{-1} \right) H \right\| \leq \frac{2\delta^2}{(\lambda + \mu)(\lambda + \mu - \delta)}, \tag{11.24}$$

for all $k \in [K]$. From the above inequalities, we have

$$\|(H_k + \mu I)^{-1}H\| \leq \|(H_k + \mu I)^{-1}H - I\| + \|I\|$$
$$\leq \|(H_k + \mu I)^{-1}H - (H + \mu I)^{-1}H\| + \|(H + \mu I)^{-1}H - I\| + \|I\|$$
$$\leq \frac{2\delta}{\lambda + \mu - \delta} + \frac{\mu}{\lambda + \mu} + 1. \tag{11.25}$$

The first and second inequality follow from triangle inequality. The third inequality follows from (11.23) and Lemma 3 in [163], which says that $\|(H+\mu I)^{-1}H - I\| = \mu/(\lambda+\mu)$. Furthermore, we also have the following bound:

$$\|\tilde{H}^{-1}H - I\| \leq \|\tilde{H}^{-1}H - (H+\mu I)^{-1}H\| + \|(H+\mu I)^{-1}H - I\|$$
$$\leq \frac{2\delta^2}{(\lambda+\mu)(\lambda+\mu-\delta)} + \frac{\mu}{\lambda+\mu}. \tag{11.26}$$

The first and second inequality follow from triangle inequality and Equation (11.24), respectively. Inequality in Equation (11.26) was earlier shown in [163]. Using inequalities in Equation (11.25) and Equation (11.26), we obtain the following bound:

$$\rho = \|\tilde{H}^{-1}H - I\| + \frac{\gamma}{K}\sum_{k=1}^{K} \|(H_k+\mu I)^{-1}H\|$$
$$\leq \left[\frac{2\delta^2}{(\lambda+\mu)(\lambda+\mu-\delta)} + \frac{\mu}{\lambda+\mu}\right] + \gamma\left[\frac{2\delta}{\lambda+\mu-\delta} + \frac{\mu}{\lambda+\mu} + 1\right]$$

Let us first consider the case of $\lambda \geq 2\sqrt{2}\delta$. In this case, we set $\mu = 0$, $\gamma = 1/8$ and hence, we have

$$\rho \leq \frac{2\delta^2}{\lambda(\lambda-\delta)} + \gamma\left[\frac{2\delta}{\lambda-\delta} + 1\right] < 0.53 < \frac{2}{3}.$$

Now, consider the case of $\lambda < 2\sqrt{2}\delta$. We use ψ to denote $8\delta/\lambda$. Note that $\psi > 2$. In this case, we set $\mu = (8\delta^2/\lambda) - \lambda$ and $\gamma = \lambda^2/(192\delta^2)$ and hence, the following holds:

$$\rho \leq \frac{2}{\psi(\psi-1)} + (1+\gamma)\left(1 - \frac{\lambda}{\psi\delta}\right) + \gamma\left(1 + \frac{2}{\psi-1}\right). \tag{11.27}$$

We observe that $\psi - 1 > \psi/2$ (since $\psi > 2$). Further, the following inequality holds:

$$(1+\gamma)\left(1 - \frac{\lambda}{\psi\delta}\right) \leq 1 - \frac{\lambda}{\psi\delta} + \gamma.$$

Substituting the above inequalities in Equation (11.27), we have the following:

$$\rho \leq \frac{4}{\psi^2} + 1 - \frac{\lambda}{\psi\delta} + \gamma + \gamma\left(1 + \frac{2}{\psi-1}\right) \leq 1 - \frac{\lambda^2}{16\delta^2} + 4\gamma \leq 1 - \frac{\lambda^2}{24\delta^2}.$$

The second inequality is due to that fact that $\psi - 1 > 1$ and the fact that $\gamma = \lambda^2/(192\delta^2)$. Hence, we have the desired result. □

Lemma 11.11.3. *Suppose a function $h : \mathbb{R}^d \to \mathbb{R}$ is L-smooth and λ-strongly convex. Let $x^* = \arg\min h(x)$. Then, we have the following:*

$$\langle \nabla h(x') - h(x), x' - x\rangle \geq \frac{1}{L}\|\nabla h(x') - \nabla h_i(x)\|^2,$$
$$\|\nabla h(x)\| \geq \lambda\|x - x^*\|,$$

for all $x', x \in \mathbb{R}^d$.

Algorithm 29: A DISTRIBUTED VERSION OF SVRG (f, x^0, s, r, h)

```
Data: $f(x) = \frac{1}{K}\sum_{k=1}^{K} F_k(x)$, initial point $x^0 \in \mathbb{R}^d$, stepsize $h > 0$
for $t = 1$ to $s$ do
    Compute $\nabla f(x^t)$ and distribute to all machines
    for $k = 1$ to $K$ do in parallel
        $x_k^t = x^t$
        for $j = 1$ to $r$ do
            Sample $i \in \mathcal{P}_k$ (e.g., uniformly at random)
            $x_k^t = x_k^t - h\left(\nabla f_i(x_k^t) - \nabla f_i(x^{t-1}) + \nabla f(x^{t-1})\right)$
        end
    end
    $x^t = \frac{1}{K}\sum_{k=1}^{K} x_k^t$
end
return $x^t$
```

11.12 A Practical Distributed Version of SVRG

In Section 11.6 we pointed out how running SVRG as a local solver in INEXACTDANE in certain setting is equivalent to the Algorithm 29 below. It is a distributed version of SVRG, that has been observed to perform well in practice [76], but has not been directly analyzed. Note that there exist another way to obtain exactly the same algorithm. That is to rewrite the local subproblem (11.2) as follows

$$g_{k,\mu}^t(x) = \sum_{i\in\mathcal{P}_k}\left[f_i(x) - \left\langle \nabla f_i(x^{t-1}) - \nabla f(x^{t-1}), x\right\rangle + \frac{\mu}{2}\|x - x^{t-1}\|^2\right],$$

and directly apply SGD locally on this reformulation within INEXACTDANE and set $\mu = 0$.

11.13 Additional Experiments

In this section, we provide extended version of what was already presented in Section 11.7, along with results of different types. In particular, we extend the comparison of INEXACTDANE, AIDE and COCOA+ to various settings and with different datasets. We also study of the effect of varying local iterations between rounds of communication on the performance of the algorithms. We present further results showing performance under varying number of nodes onto which the dataset is distributed, and highlight a case of non-random data distribution, in which DANE diverges, while the performance of INEXACTDANE degrades only slightly, compared to benchmark with random data distribution.

Again, the following default statements are true, unless stated otherwise. We use SVRG as local solver for INEXACTDANE and AIDE, and we use SDCA in COCOA+. We run all presented methods for a single pass over data available locally in every iteration. We partition data randomly (mimicking the iid data distribution) across 8 nodes. We present the best performance selected from a range of stepsize parameters for SVRG and aggregation parameters for COCOA+. In the plots, we use DANE as label for INEXACTDANE.

We omit the experiments for quadratic objectives as the results were very similar to the ones presented here and hence, did not provide any additional insights, compared to the experiments with classification on publicly available datasets.

11.13.1 INEXACTDANE, AIDE and COCOA+

In Figures 11.2 and 11.3, we present a comparison of INEXACTDANE, AIDE and COCOA+ on a binary classification task on the rcv1 dataset. Plots in top row are for logistic loss, and bottom is for smoothed hinge loss. We apply L2-regularization with λ set to $1/(cN)$ for $c \in \{1, 10, 100\}$, which correspond to left, middle and right column respectively. In Figure 11.2 we partition the data randomly to 8 nodes, while in Figure 11.3 we use partitioning across 64 nodes. To strengthen our claims, in Figures 11.4 and 11.5, we provide experiment with settings analogous to the ones in Figure 11.2, but on covtype and realsim datasets respectively.

In this experiment, we can see a common pattern arising in different settings. The benefit of acceleration in AIDE is present only when the condition number $\kappa := L/\mu > N$, and the larger it is, the larger is the gap in performance. This is to be expected, as the acceleration of the fast stochastic methods changes κ in convergence rates to $\sqrt{N\kappa}$ [154]. Both INEXACTDANE and AIDE outperform COCOA+, with AIDE performing much better in all studied settings. The behaviour of COCOA+, where if one looks only at suboptimality of primal function value, the algorithm quickly converges to modest accuracy, and then converges very slowly to higher accuracies, has been confirmed as correct by its authors.

11.13.2 Varying amount of local computation

In Figure 11.6 we demonstrate how spending more local computational resources in each iteration to solve the local subproblem (11.2) leads to faster convergence in terms of number of iterations. Note that in the settings presented, it is not possible to get close form solution to the subproblems, and hence we can only approximate behaviour of DANE by running a local method for a long time. In number of cases in the above, running SVRG for 4 passes through local data already provides little to none improvement. The only dataset on which significant improvement is visible is covtype. This behaviour likely is due to the fact that $N \gg d$, and hence under random data distribution, the local problems can be seen as δ-related with very small δ.

The labels in the figure mean represent the following: The labels DANE/6 and DANE/3 correspond to running the local SVRG algorithm for one-sixth and one-third of pass through local data in every iteration. The labels DANE*2 and DANE*4 corre-

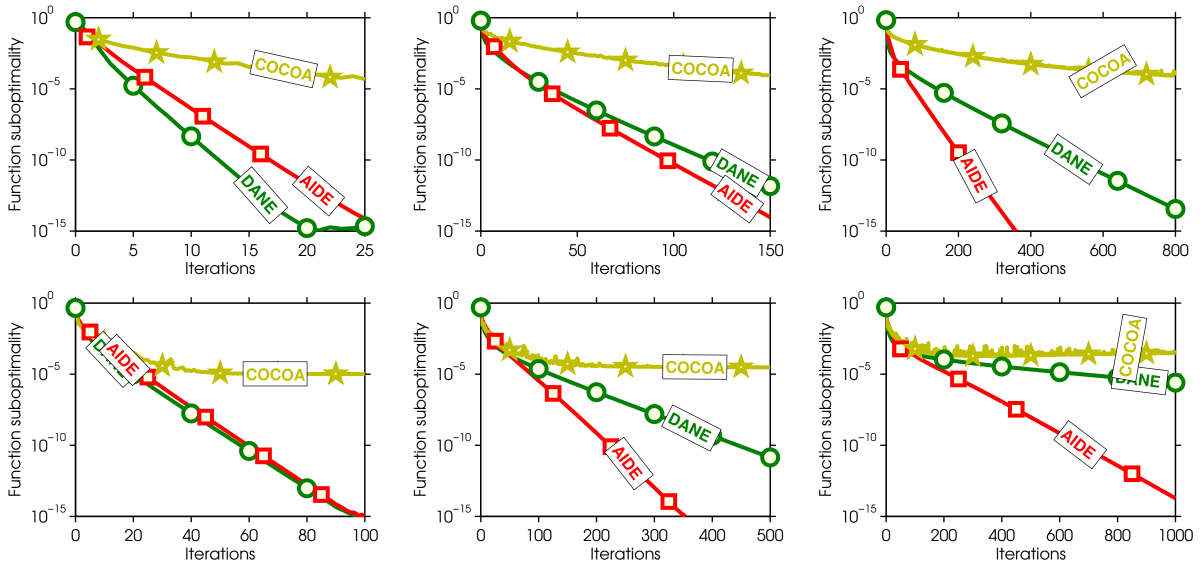


Figure 11.2: rcv1 dataset, 8 nodes, regularization parameter λ set to $1/(cN)$, for $c \in \{1, 10, 100\}$ (in left/middle/right columns respectively). Top row: logistic loss. Bottom row: smoothed hinge loss.

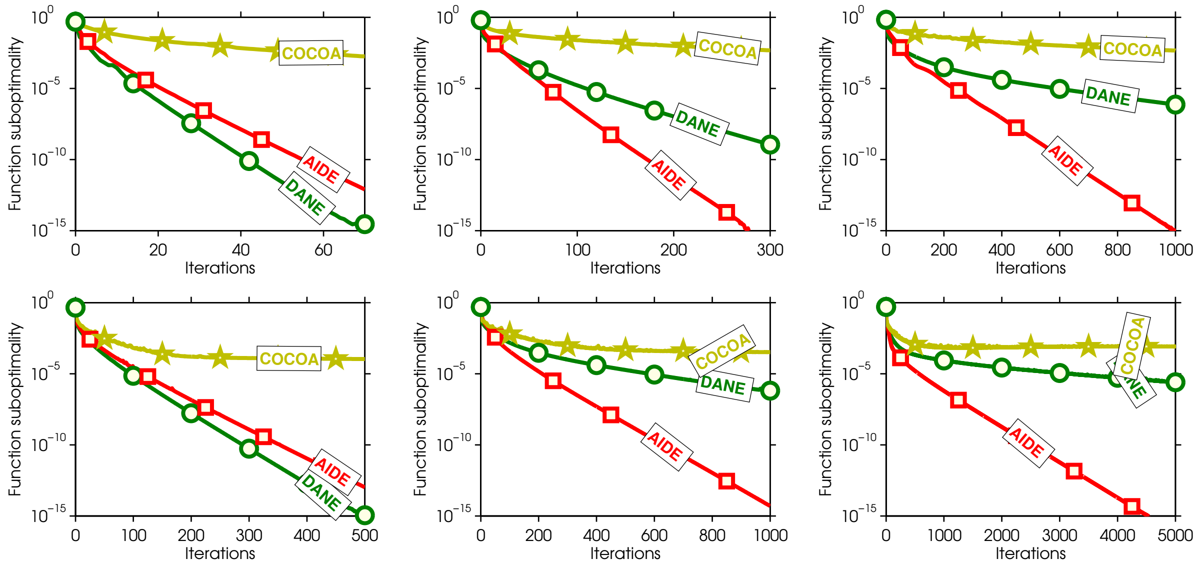


Figure 11.3: rcv1 dataset, 64 nodes, regularization parameter λ set to $1/(cN)$, for $c \in \{1, 10, 100\}$ (in left/middle/right columns respectively). Top row: logistic loss. Bottom row: smoothed hinge loss.

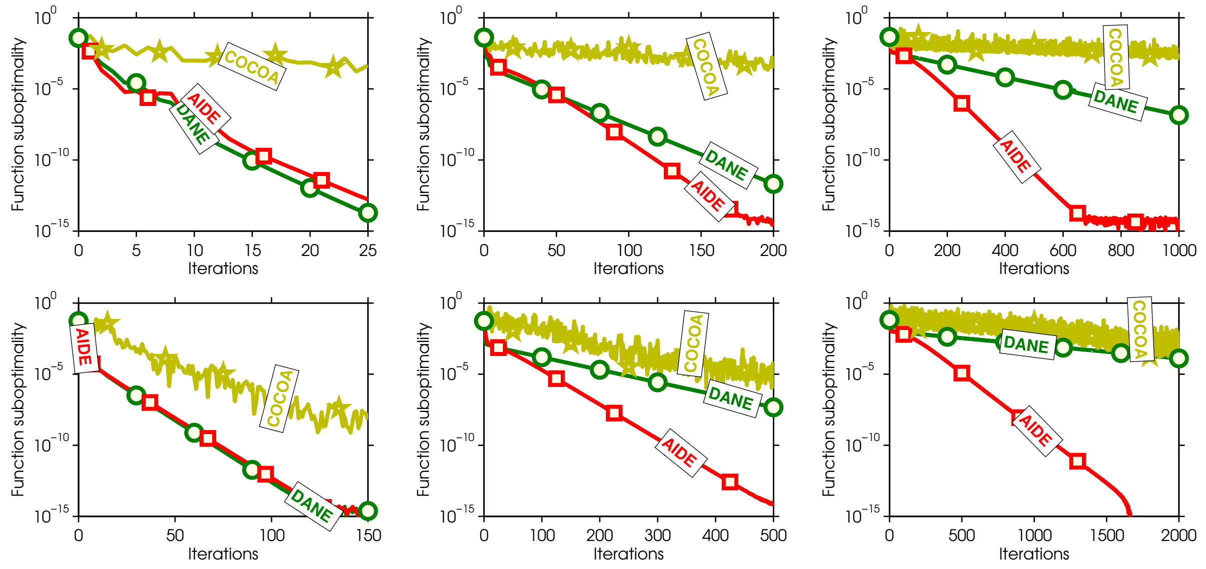


Figure 11.4: covtype dataset, 8 nodes, regularization parameter λ set to $1/(cN)$, for $c \in \{1, 10, 100\}$ (in left/middle/right columns respectively). Top row: logistic loss. Bottom row: smoothed hinge loss.

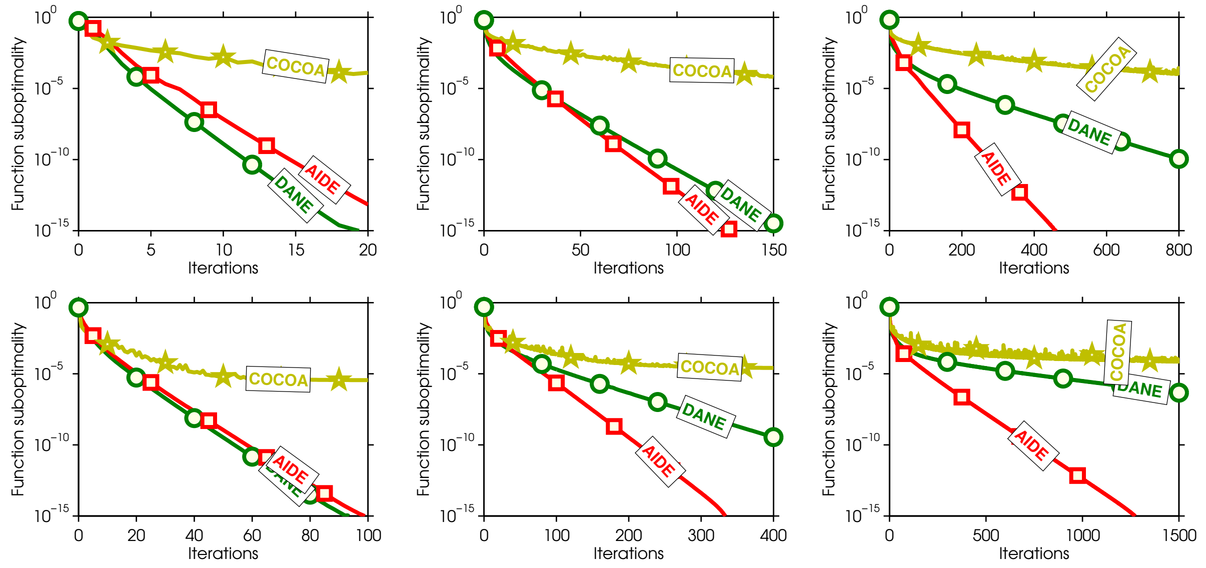


Figure 11.5: realsim dataset, 8 nodes, regularization parameter λ set to $1/(cN)$, for $c \in \{1, 10, 100\}$ (in left/middle/right columns respectively). Top row: logistic loss. Bottom row: smoothed hinge loss.

spond to running the local SVRG algorithm for two and four passes through the local data.

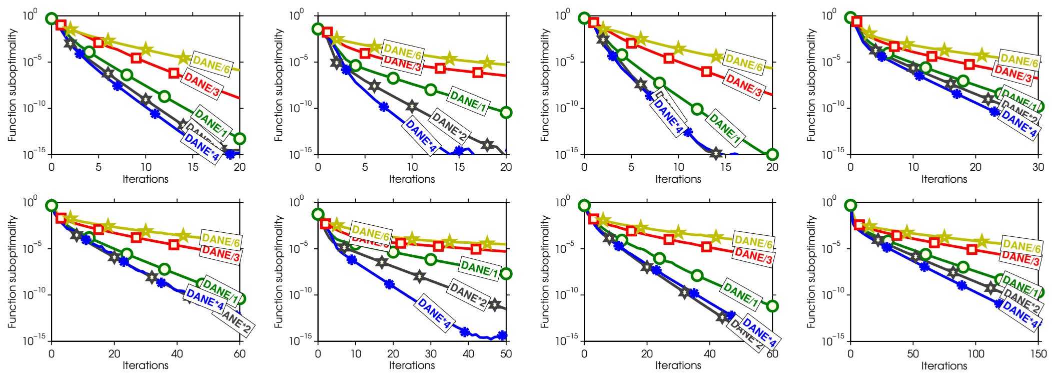


Figure 11.6: Varying number of passes through local data per iteration in range $[1/6, 4]$. Top row: logistic loss, Bottom row: smoothed hinge loss. Datasets in columns: rcv1, covtype, realsim, url

11.13.3 Node scaling

The plots in top row in Figure 11.7 show how the performance changes, as we partition the data across different number of nodes, but keep the local work equal to one pass through local data. The performance degrades as we increase number of nodes, because we do less relative work on any single computer. However, this is a positive result, as the algorithm *always* converges.

In the bottom row, we double the number of passes over local data when we double the number of nodes. This is motivated to compare how the algorithm performs, when equal number of iteration of local SVRG is used. In most cases, the algorithms perform very similarly, demonstrating the robustness of INEXACTDANE. For rcv1 and realsim datasets partitioned on 64 nodes, the convergence slows down. In this case, the optimal stepsize for local SVRG was significantly smaller than in the other cases. This was likely caused by getting into region where the aggregation starts to be unstable, and thus higher accuracy on local subproblems leads to worse overall performance.

11.13.4 Inconvenient data partitioning

In the last experiment, we depart from theory, and observe that INEXACTDANE is, compared to DANE, much more robust to arbitrary partitions of the data. In particular, in Figure 11.8 we compare INEXACTDANE in two settings: *random*, which corresponds to random data partition to 2 nodes, and *output*, where the data are partitioned according to their output label — positive examples on one node, negative examples on the other. In this setting, DANE diverges, while the performance of INEXACTDANE drops down only slightly. We observed that the optimal stepsize for local SVRG is about $5 - 10\%$ smaller in the *output* case, compared to the *random* case.

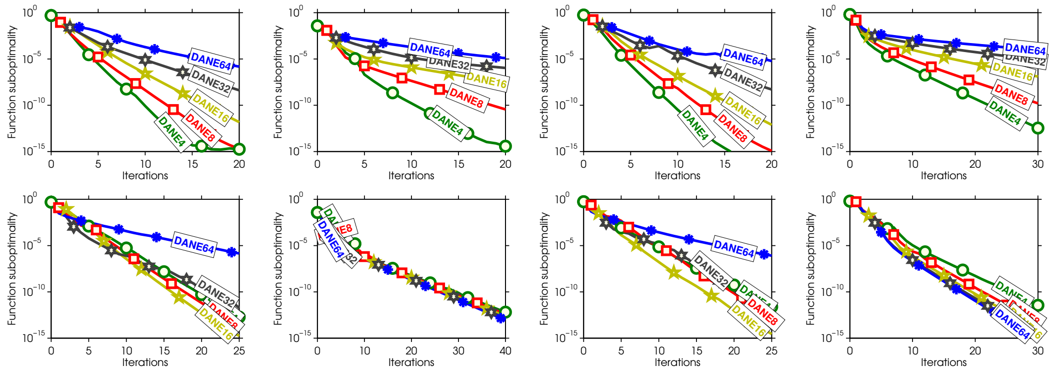


Figure 11.7: Performance comparison as data is partitioned across 4–64 nodes, with logistic loss. Top row: Single pass through local data per iteration. Bottom row: Fixed number of updates of local SVRG per iteration. Datasets in columns: rcv1, covtype, realsim, url.

This observation is particularly appealing for practitioners, as the data in huge scale applications are often partitioned "as is", i.e., it is given and one does not have the opportunity to randomly reshuffle the data between nodes.

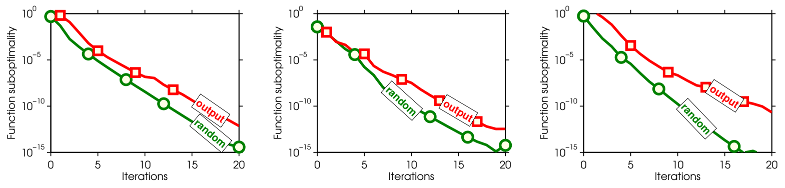


Figure 11.8: Data partitioned randomly, vs. data partitioned according to their output label. Datasets in columns: rcv1, covtype and realsim.

Practical considerations: Although the experiments in Section 11.13.1 suggest superiority of AIDE, it may not always be the case in practice. AIDE comes with the additional requirement to set the catalyst acceleration parameter τ. At the moment, there is not any simple rule guiding its choice, as the optimal τ depends on properties of the algorithm being accelerated, which are usually not known — see Section 3.1 of [88] for further details. In contrast, COCOA+ is usually a slightly easier to tune in practice, since with SDCA one can use data-independent aggregation parameter equal to $1/K$, and effectively have hyper-parameter-free algorithm. In the case of INEXACTDANE, natural local solvers would require a choice of hyper-parameter such as stepsize. While this can often be affordable to compute, it is not data-independent.

Chapter 12

My Other Research — In a Nutshell

Besides optimization methods for machine learning applications, I have also worked on a number of other topics such as kernel methods, hypothesis testing, dependence measures, functional regression, *which are not part of my thesis*. Here, I briefly describe my contributions that form the core of my research in these areas.

Kernel Methods, Dependence Measures & Hypothesis Testing. Measuring dependencies and conditional dependencies are of great importance in many scientific fields including machine learning, and statistics. There are numerous problems where we want to know how large the dependence is between random variables, and how this dependence changes if we observe other random variables. In an ICML'13 paper [130], we developed new kernel based dependence measures with scale invariance property and showed the effectiveness of our dependence measure on real-world problems. I also worked on kernel based two sample testing problem. In particular, in a series of papers [126, 136, 137] we proved (through theoretical and empirical results) that MMD based two sample testing — a kernel based two sample test — also suffers from the curse of dimensionality. Such a result was important because there was a wide misconception that these measures do not suffer from the curse of dimensionality and hence, also work for high dimension problems. Our papers cleared this misconception and provided first theoretical results for the power of the MMD based two sample hypothesis tests.

Machine Learning on Functional Data. Another important aspect of many modern machine learning applications is the structure in the input data. Modern data collection techniques motivate settings where the training data is no longer of simple form such as features. For example, a Facebook user profile contains very rich information about the user such as posts, friends list, likes, photos, polls, etc. Unfortunately, most of the existing machine learning and statistical techniques cannot handle such data, often resorting to ad-hoc approaches, thereby ignoring the underlying rich structure in the data. This necessitates the development of a different machine learning paradigm where the true structure in the complex data can be exploited. In a UAI'14 [131], we developed a simple nearest neighbor based algorithm for handling data of various forms. In fact, we considered a strictly generalized scenario of noisy and incomplete/missing data. We analyzed the theoretical properties of our proposed estimator and demonstrated its performance through practical experiments. We also proposed an approach to choose the number of nearest neighbors to be used, thereby alleviating the problem of cross validation in nearest neighbor based algorithms.

Bibliography

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `http://tensorflow.org/`. (pages 2, 67, and 221)

[2] A. Agarwal and L. Bottou. A lower bound for the optimization of finite sums. *CoRR*, abs/1410.0723, 2014. (pages 4, 18, 46, 61, 62, 82, 83, 112, and 150)

[3] A. Agarwal and J. C. Duchi. Distributed delayed stochastic optimization. In *Proceedings of the 25th Annual Conference on Neural Information Processing Systems (NIPS)*, pages 873–881, 2011. (pages 46, 149, and 150)

[4] N. Agarwal, B. Bullins, and E. Hazan. Second order stochastic optimization in linear time. *CoRR*, abs/1602.03943, 2016. (page 62)

[5] N. Agarwal, A. Z. Zeyuan, B. Bullins, E. Hazan, and T. Ma. Finding approximate local minima for nonconvex optimization in linear time. *CoRR*, abs/1611.01146, 2016. (pages 60, 61, and 66)

[6] P. K. Agarwal, S. Har-Peled, and K. R. Varadarajan. Geometric approximation via coresets. In *Combinatorial and Computational Geometry, MSRI*, pages 1–30. University Press, 2005. (page 207)

[7] S. Ahn, A. Korattikara, and M. Welling. Bayesian posterior sampling via stochastic gradient Fisher scoring. In *Proceedings of the 29th International Conference on Machine Learning (ICML)*, 2012. (pages 129 and 139)

[8] S. Ahn, B. Shahbaba, and M. Welling. Distributed stochastic gradient MCMC. In *Proceedings of the 31st International Conference on Machine Learning (ICML)*, 2014. (pages 129 and 139)

[9] A. Antoniadis, I. Gijbels, and M. Nikolova. Penalized likelihood regression for generalized linear models with non-quadratic penalties. *Annals of the Institute of Statistical Mathematics*, 63(3):585–615, June 2009. (page 54)

[10] A. Auslender. *Optimisation Méthodes Numériques*. Masson, 1976. (page 181)

[11] F. Bach, R. Jenatton, J. Mairal, and G. Obozinski. Convex optimization with sparsity-inducing norms. In S. Sra, S. Nowozin, and S. J. Wright, editors, *Optimization for Machine Learning*. MIT Press, 2011. (page 81)

[12] M. F. Balcan, S. Ehrlich, and Y. Liang. Distributed k-means and k-median clustering on general communication topologies. In *Proceedings of the 27th Annual Conference on Neural Information Processing Systems (NIPS)*, pages 1995–2003, 2013. (page 207)

[13] A. Beck and L. Tetruashvili. On the convergence of block coordinate descent type methods. *SIAM Journal on Optimization*, 23(4):2037–2060, 2013. (page 183)

[14] P. Berman, N. Kovoor, and P. M. Pardalos. A linear-time algorithm for the least-distance problem. Technical report, Pennsylvania State University, Department of Computer Science, 1992. (page 191)

[15] D. Bertsekas and J. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods*. Prentice-Hall, 1989. (page 46)

[16] D. P. Bertsekas. *Nonlinear Programming*. Athena Scientific, 1995. (page 110)

[17] D. P. Bertsekas. *Nonlinear Programming*. Athena Scientific, Belmont, MA, second edition, 1999. (page 181)

[18] D. P. Bertsekas. Incremental gradient, subgradient, and proximal methods for convex optimization: A survey. *Optimization for Machine Learning*, 2010:1–38, 2011. (pages 18, 45, 46, and 150)

[19] L. Bottou. Stochastic gradient learning in neural networks. *Proceedings of Neuro-Nımes*, 91(8), 1991. (pages 19, 61, and 83)

[20] S. Boyd. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine Learning*, 3(1): 1–122, 2010. (page 207)

[21] J.K. Bradley, A. Kyrola, D. Bickson, and C. Guestrin. Parallel coordinate descent for L1-regularized loss minimization. In L. Getoor and T. Scheffer, editors, *Proceedings of the 28th International Conference on Machine Learning*, pages 321–328. Omnipress, 2011. (page 182)

[22] Y. Carmon, J. C. Duchi, O. Hinder, and A. Sidford. Accelerated methods for non-convex optimization. *CoRR*, abs/1611.00756, 2016. (pages 60, 61, 66, 68, 78, and 79)

[23] C. Cartis and K. Scheinberg. Global convergence rate analysis of unconstrained optimization methods based on probabilistic models. *Mathematical Programming*, pages 1–39, 2017. (page 77)

[24] C. Chang and C. Lin. Libsvm: A library for support vector machines. *ACM Trans. Intell. Syst. Technol.*, 2(3):27:1–27:27, May 2011. ISSN 2157-6904. (page 194)

[25] C. Chen, N. Ding, and L. Carin. On the convergence of stochastic gradient mcmc algorithms with high-order integrators. In *Proceedings of the 29th Annual Conference on Neural Information Processing Systems (NIPS)*, 2015. (pages 131, 135, 140, and 142)

[26] T. Chen, E. B. Fox, and C. Guestrin. Stochastic gradient Hamiltonian Monte Carlo. In *Proceedings of the 31st International Conference on Machine Learning (ICML)*, 2014. (page 130)

[27] A. Choromanska, M. Henaff, M. Mathieu, G. B. Arous, and Y. LeCun. The loss surface of multilayer networks. *CoRR*, abs/1412.0233, 2014. (pages 7 and 60)

[28] M. Collins, A. Globerson, T. Koo, X. Carreras, and P. L. Bartlett. Exponentiated gradient algorithms for conditional random fields and max-margin markov networks. *Journal of Machine Learning Research (JMLR)*, 9:1775–1822, 2008. (pages 9 and 109)

[29] Y. Dauphin, H. de Vries, and Y. Bengio. Equilibrated adaptive learning rates for non-convex optimization. In *Proceedings of the 29th Annual Conference on Neural Information Processing Systems (NIPS)*, pages 1504–1512, 2015. (pages 7 and 60)

[30] Y. N. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *Proceedings of the 28th Annual Conference on Neural Information Processing Systems (NIPS)*, pages 2933–2941, 2014. (pages 7 and 60)

[31] A. Defazio. *New Optimization Methods for Machine Learning*. PhD thesis, Australian National University, 2014. (page 150)

[32] A. Defazio. A simple practical accelerated method for finite sums. pages 676–684, 2016. (page 229)

[33] A. Defazio, F. Bach, and S. Lacoste-Julien. SAGA: A fast incremental gradient method with support for non-strongly convex composite objectives. In *Proceedings of the 28th Annual Conference on Neural Information Processing Systems (NIPS)*, pages 1646–1654. 2014. (pages 17, 18, 45, 46, 48, 50, 61, 81, 83, 87, 92, 110, 111, 116, 119, 130, 132, 133, 136, 149, 150, 151, 154, 158, 160, 163, 168, 174, and 229)

[34] A. J. Defazio, T. S. Caetano, and J. Domke. Finito: A faster, permutable incremental gradient method for big data problems. pages 1125–1133, 2014. (pages 18, 61, 149, and 150)

[35] O. Dekel, R. Gilad-Bachrach, O. Shamir, and L. Xiao. Optimal distributed online prediction using mini-batches. *Journal of Machine Learning Research*, 13(1):165–202, 2012. (pages 26 and 149)

[36] L. Deng and D. Yu. Deep learning: Methods and applications. *Foundations and Trends Signal Processing*, 7:197–387, 2014. (page 45)

[37] N. Ding, Y. Fang, R. Babbush, C. Chen, R. D. Skeel, and H. Neven. Bayesian sampling using stochastic gradient thermostats. In *Proceedings of the 28th Annual Conference on Neural Information Processing Systems (NIPS)*, 2014. (pages 130 and 139)

[38] D. Feldman and M. Langberg. A unified framework for approximating and clustering data. In *Proceedings of the Forty-third Annual ACM Symposium on Theory of Computing (STOC)*, pages 569–578, 2011. (pages 12, 206, 207, 208, and 219)

[39] D. Feldman, M. Schmidt, and C. Sohler. Turning big data into tiny data: Constant-

size coresets for k-means, pca and projective clustering. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1434–1453, 2013. (pages 12, 206, 207, and 219)

[40] M. Frank and P. Wolfe. An algorithm for quadratic programming. *Naval Research Logistics Quarterly*, 3(1-2):95–110, March 1956. (pages 109, 110, and 112)

[41] J. Friedman, T. Hastie, H. Höfling, R. Tibshirani, et al. Pathwise coordinate optimization. *The Annals of Applied Statistics*, 1(2):302–332, 2007. (page 181)

[42] S. Fujishige and S. Isotani. A submodular function minimization algorithm based on the minimum-norm base. *Pacific Journal of Optimization*, 7(1):3–17, 2011. (pages 9 and 109)

[43] M. Fukushima and H. Mine. A generalized proximal point algorithm for certain non-convex minimization problems. *International Journal of Systems Science*, 12(8): 989–1000, 1981. (page 82)

[44] D. Gabay and B. Mercier. A dual algorithm for the solution of nonlinear variational problems via finite element approximation. *Computers & Mathematics with Applications*, 2(1):17–40, 1976. (page 182)

[45] D. Garber and E. Hazan. Faster rates for the frank-wolfe method over strongly-convex sets. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, pages 541–549, 2015. (page 110)

[46] R. Ge, F. Huang, C. Jin, and Y. Yuan. Escaping from saddle points - online stochastic gradient for tensor decomposition. In *Proceedings of The 28th Conference on Learning Theory, COLT*, pages 797–842, 2015. (pages 30, 60, 61, 66, and 70)

[47] S. Ghadimi and G. Lan. Stochastic first- and zeroth-order methods for nonconvex stochastic programming. *SIAM Journal on Optimization*, 23(4):2341–2368, 2013. (pages 4, 5, 19, 20, 31, 46, 47, 59, 61, 62, 83, 84, 109, 110, 111, and 124)

[48] S. Ghadimi, G. Lan, and H. Zhang. Mini-batch stochastic approximation methods for nonconvex stochastic composite optimization. *Mathematical Programming*, 155 (1-2):267–305, 2014. (pages 8, 82, 84, 109, and 110)

[49] M. Girolami and B. Calderhead. Riemann manifold Langevin and Hamiltonian Monte Carlo methods. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 2011. (page 130)

[50] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2010. (page 29)

[51] R. Glowinski and A. Marrocco. Sur l'approximation, par éléments finis d'ordre un, et la résolution, par pénalisation-dualité d'une classe de problèmes de dirichlet non linéares. *Revue Française d'Automatique, Informatique, et Recherche Opérationelle*, 1975. (page 182)

[52] G. H. Golub and C. F. Van Loan. *Matrix Computations*. John Hopkins University Press, Baltimore, MD, 3rd edition, 1996. (page 182)

[53] M. Gürbüzbalaban, A. Ozdaglar, and P. Parrilo. A globally convergent incremental Newton method. *Mathematical Programming*, 151(1):283–313, 2015. (page 149)

[54] Z. Harchaoui, A. Juditsky, and A. Nemirovski. Conditional gradient algorithms for norm-regularized smooth convex optimization. *Mathematical Programming*, 152(1-2):75–112, apr 2014. (page 109)

[55] E. Hazan and S. Kale. Projection-free online learning. In *Proceedings of the 29th International Conference on Machine Learning (ICML)*, pages 1843–1850, 2012. (pages 109, 116, and 124)

[56] E. Hazan and H. Luo. Variance-reduced and projection-free stochastic optimization. *CoRR*, abs/1602.02101, 2016. (pages 109, 110, 111, 113, 116, and 124)

[57] E. Hazan, K. Levy, and S. Shalev-Shwartz. Beyond convexity: Stochastic quasi-convex optimization. In *Proceedings of the 29th Annual Conference on Neural Information Processing Systems (NIPS)*, pages 1585–1593, 2015. (page 19)

[58] A. Hefny, S. J. Reddi, and S. Sra. Coordinate descent algorithms with coupling constraints: Lessons learned. In *NIPS Workshop on Software Engineering For Machine Learning*, 2014. (page 193)

[59] G. E. Hinton and R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006. (page 69)

[60] M. Hong. A distributed, asynchronous and incremental algorithm for nonconvex optimization: An admm based approach. *CoRR*, abs/1412.6058, 2014. (pages 19 and 46)

[61] M. Hong and Z. Luo. On the linear convergence of the alternating direction method of multipliers. *CoRR*, abs/1208.3922, 2012. (page 183)

[62] M. Hong, X. Wang, M. Razaviyayn, and Z.-Q. Luo. Iteration Complexity Analysis of Block Coordinate Descent Methods. *CoRR*, abs/1310.6957, 2013. (page 181)

[63] C. J. Hsieh and I. S. Dhillon. Fast coordinate descent methods with variable selection for non-negative matrix factorization. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining(KDD)*, pages 1064–1072, 2011. (page 181)

[64] C. J. Hsieh, K. W. Chang, C. J. Lin, S. S. Keerthi, and S. Sundararajan. A dual coordinate descent method for large-scale linear SVM. In *Proceedings of the 25th International Conference on Machine Learning (ICML)*, pages 408–415, 2008. (pages 181 and 182)

[65] C. J. Hsieh, M. A. Sustik, I. S. Dhillon, and P. D. Ravikumar. Sparse inverse covariance matrix estimation using quadratic approximation. In *Proceedings of the 25th Annual Conference on Neural Information Processing Systems (NIPS)*, pages 2330–2338, 2011. (page 181)

[66] M. Jaggi. Revisiting Frank-Wolfe: Projection-free sparse convex optimization. In *Proceedings of the 30tt International Conference on Machine Learning (ICML)*, pages 427–435, 2013. (pages 109 and 110)

[67] M. Jaggi, V. Smith, M. Takac, J. Terhorst, S. Krishnan, T. Hofmann, and Michael I. Jordan. Communication-Efficient Distributed Dual Coordinate Ascent. In *Proceedings of the 28th Annual Conference on Neural Information Processing Systems (NIPS)*, pages 3068–3076. 2014. (pages 12 and 222)

[68] G. James, D. Witten, T. Hastie, and R. Tibshirani. *An Introduction to Statistical Learning: with Applications in R*. Springer Texts in Statistics. Springer, 2013. (page 45)

[69] C. Jin, R. Ge, P. Netrapalli, S. M. Kakade, and M. I. Jordan. How to escape saddle points efficiently. *CoRR*, abs/1703.00887, 2017. (pages 61 and 66)

[70] T. Joachims. Training linear svms in linear time. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, pages 217–226. ACM, 2006. ISBN 1-59593-339-5. (page 207)

[71] R. Johnson and T. Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In *Proceedings of the 27th Annual Conference on Neural Information Processing Systems (NIPS)*, pages 315–323. 2013. (pages 17, 18, 21, 24, 29, 43, 46, 61, 64, 65, 83, 85, 110, 116, 130, 132, 134, 149, 150, 151, 153, 154, 159, 160, 168, 179, 207, 218, 219, 222, 223, and 229)

[72] H. Karimi, J. Nutini, and M. W. Schmidt. Linear convergence of gradient and proximal-gradient methods under the polyak-łojasiewicz condition. In *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD*, pages 795–811, 2016. (pages 83, 90, and 91)

[73] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. (pages 67 and 77)

[74] J. Konečný and P. Richtárik. Semi-Stochastic Gradient Descent Methods. *CoRR*, abs/1312.1666, 2013. (pages 21, 149, 150, 222, 223, and 229)

[75] J. Konečný, J. Liu, P. Richtárik, and M. Takáč. Mini-Batch Semi-Stochastic Gradient Descent in the Proximal Setting. *arXiv:1504.04407*, 2015. (pages 18, 61, 149, and 150)

[76] J. Konečný, B. McMahan, and D. Ramage. Federated optimization: distributed optimization beyond the datacenter. *CoRR*, abs/1511.03575, 2015. (pages 222, 229, 231, and 242)

[77] H. J. Kushner and D. S. Clark. *Stochastic approximation methods for constrained and unconstrained systems*, volume 26. Springer Science & Business Media, 2012. (pages 19, 46, and 61)

[78] S. Lacoste-Julien. Convergence Rate of Frank-Wolfe for Non-Convex Objectives. abs/1607.00345, 2016. (pages xiv, 110, 112, 113, and 124)

[79] S. Lacoste-Julien and M. Jaggi. On the global linear convergence of frank-wolfe optimization variants. In *Proceedings of the 29th Annual Conference on Neural Information Processing Systems (NIPS)*, pages 496–504, 2015. (page 110)

[80] S. Lacoste-Julien, M. Jaggi, M. Schmidt, and P. Pletscher. Block-coordinate frank-

wolfe optimization for structural svms. *CoRR*, abs/1207.4747, 2012. (page 183)

[81] G. Lan and Y. Zhou. An optimal randomized incremental gradient method. *CoRR*, abs/1507.02000, 2015. (pages 18, 61, and 83)

[82] C. L. Lawson and R. J. Hanson. *Solving Least Squares Problems*. Prentice–Hall, Englewood Cliffs, NJ, 1974. Reissued with a survey on recent developments by SIAM, Philadelphia, 1995. (page 182)

[83] K. Y. Levy. The power of normalization: Faster evasion of saddle points. *CoRR*, abs/1611.04831, 2016. (page 61)

[84] M. Li, D. Andersen, A. Smola, and K. Yu. Communication efficient distributed machine learning with the parameter server. In *Proceedings of the 28th Annual Conference on Neural Information Processing Systems (NIPS)*, pages 19–27. 2014. (pages 2, 149, 208, and 221)

[85] M. Li, T. Zhang, Y. Chen, and A. J. Smola. Efficient mini-batch training for stochastic optimization. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, pages 661–670. ACM, 2014. (page 26)

[86] X. Li, T. Zhao, R. Arora, H. Liu, and J. Haupt. Stochastic variance reduced optimization for nonconvex sparse learning. In *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, 2016. (page 84)

[87] X. Lian, Y. Huang, Y. Li, and J. Liu. Asynchronous Parallel Stochastic Gradient for Nonconvex Optimization. In *Proceedings of the 29th Annual Conference on Neural Information Processing Systems (NIPS)*, 2015. (pages 19 and 61)

[88] H. Lin, J. Mairal, and Z. Harchaoui. A universal catalyst for first-order optimization. In *Proceedings of the 29th Annual Conference on Neural Information Processing Systems (NIPS)*, pages 3366–3374. 2015. (pages 227, 230, 238, 239, and 247)

[89] J. Liu and S. J. Wright. Asynchronous stochastic coordinate descent: Parallelism and convergence properties. *SIAM Journal on Optimization*, 25(1):351–376, January 2015. (page 91)

[90] J. Liu and S. J. Wright. Asynchronous stochastic coordinate descent: Parallelism and convergence properties. *SIAM Journal on Optimization*, 25(1):351–376, 2015. (page 150)

[91] J. Liu, S. J. Wright, C. Ré, V. Bittorf, and S. Sridhar. An asynchronous parallel stochastic coordinate descent algorithm. *CoRR*, abs/1311.1873, 2013. (pages 182, 183, 186, and 188)

[92] J. Liu, S. J. Wright, C. Ré, V. Bittorf, and S. Sridhar. An asynchronous parallel stochastic coordinate descent algorithm. In *Proceedings of the 31st International Conference on Machine Learning (ICML)*, pages 469–477, 2014. (pages 150 and 154)

[93] L. Ljung. Analysis of recursive stochastic algorithms. *IEEE Transactions on Automatic Control*, 22(4):551–575, 1977. (pages 19 and 61)

[94] Z. Luo and P. Tseng. On the convergence of the coordinate descent method for

convex differentiable minimization. *Journal of Optimization Theory and Applications*, 72(1):7–35, 1992. (page 183)

[95] C. Ma, J. Konečný, M. Jaggi, V. Smith, M. I. Jordan, P. Richtárik, and M. Takáč. Distributed optimization with arbitrary local solvers. *CoRR*, abs/1512.04039, 2015. (pages 222 and 229)

[96] C. Ma, V. Smith, M. Jaggi, M. I. Jordan, P. Richtárik, and M. Takáč. Adding vs. averaging in distributed primal-dual optimization. *CoRR*, abs/1502.03508, 2015. (page 222)

[97] Y. Ma, T. Chen, and E. Fox. A complete recipe for stochastic gradient MCMC. In *Proceedings of the 29th Annual Conference on Neural Information Processing Systems (NIPS)*, 2015. (pages 129 and 139)

[98] D. Mahajan, S. S. Keerthi, S. Sundararajan, and L. Bottou. A functional approximation based distributed learning algorithm. *CoRR*, abs/1310.8418, 2013. (page 207)

[99] J. Mairal. Optimization with first-order surrogate functions. *CoRR*, abs/1305.3120, 2013. (page 150)

[100] J. Martens. Deep learning via hessian-free optimization. In *Proceedings of the 27th International Conference on Machine Learning (ICML)*, pages 735–742, 2010. (page 69)

[101] J. Martens and R. Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, pages 2408–2417, 2015. (page 69)

[102] H. Mine and M. Fukushima. A minimization method for the sum of a convex function and a continuously differentiable function. *Journal of Optimization Theory and Applications*, 33(1):9–23, 1981. (page 82)

[103] J. J. Moreau. Fonctions convexes duales et points proximaux dans un espace hilbertien. *C. R. Acad. Sci. Paris Sér. A Math.*, 255:2897–2899, 1962. (page 81)

[104] R. Neal. Mcmc using hamiltonian dynamics. In *Handbook of Markov Chain Monte Carlo*, 2010. (pages 9 and 129)

[105] I. Necoara and A. Patrascu. A random coordinate descent algorithm for optimization problems with composite objective function and linear coupled constraints. *Computational Optimization and Applications*, 57(2):307–337, 2014. (pages 11, 182, 183, 188, 190, 191, 194, 200, and 201)

[106] I Necoara, Y Nesterov, and F Glineur. A random coordinate descent method on large optimization problems with linear constraints. Technical report, Technical Report, University Politehnica Bucharest, 2011, 2011. (pages 11, 182, 187, 188, and 191)

[107] A. Nedić, D. P. Bertsekas, and V. S. Borkar. Distributed asynchronous incremental subgradient methods. *Studies in Computational Mathematics*, 8:381–407, 2001. (page 150)

[108] A. Nemirovski and D. Yudin. *Problem Complexity and Method Efficiency in Optimization*. John Wiley and Sons, 1983. (pages 21, 83, 104, and 111)

[109] A. Nemirovski, A. Juditsky, G. Lan, and A. Shapiro. Robust stochastic approximation approach to stochastic programming. *SIAM Journal on Optimization*, 19(4): 1574–1609, January 2009. ISSN 1052-6234. (page 197)

[110] 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. (pages 17, 18, 46, and 149)

[111] Y. Nesterov. *Introductory Lectures On Convex Optimization: A Basic Course*. Springer, 2003. (pages 4, 19, 46, 47, 62, 84, 124, 135, 206, 211, and 223)

[112] Y. Nesterov. Efficiency of coordinate descent methods on huge-scale optimization problems. Core discussion papers, Université catholique de Louvain, Center for Operations Research and Econometrics (CORE), 2010. (pages 182 and 183)

[113] Y. Nesterov. Efficiency of coordinate descent methods on huge-scale optimization problems. *SIAM Journal on Optimization*, 22(2):341–362, 2012. (pages 81 and 150)

[114] Y. Nesterov. Efficiency of coordinate descent methods on huge-scale optimization problems. *SIAM Journal on Optimization*, 22(2):341–362, 2012. (page 182)

[115] Y. Nesterov and B. T. Polyak. Cubic regularization of newton method and its global performance. *Mathematical Programming*, 108(1):177–205, 2006. (pages 3, 7, 19, 47, 60, 61, 62, 66, 67, and 76)

[116] A. Nitanda. Stochastic Proximal Gradient Descent with Acceleration Techniques. In *Proceedings of the 28th Annual Conference on Neural Information Processing Systems (NIPS)*, pages 1574–1582, 2014. (page 150)

[117] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer series in operations research and financial engineering. Springer, 1999. ISBN 9780387987934. (page 207)

[118] N. Parikh and S. Boyd. Proximal algorithms. *Foundations and Trends in Optimization*, 1(3):127–239, 2014. ISSN 2167-3888. (page 81)

[119] S. Patterson and Y. W. Teh. Stochastic gradient Riemannian Langevin dynamics on the probability simplex. In *Proceedings of the 27th Annual Conference on Neural Information Processing Systems (NIPS)*, 2013. (pages 130 and 139)

[120] B. A. Pearlmutter. Fast exact multiplication by the hessian. *Neural Computation*, 6 (1):147–160, January 1994. ISSN 0899-7667. (pages 62 and 67)

[121] J. C. Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. Technical report, Advances in Kernel Methods - Support Vector Learning, 1998. (page 182)

[122] B. T. Polyak. Gradient methods for the minimisation of functionals. *USSR Computational Mathematics and Mathematical Physics*, 3(4):864–878, January 1963. (pages 3, 19, 24, 47, 52, and 90)

[123] B. T. Polyak. *Introduction to Optimization*. Optimization Software Inc., 1987. Nov 2010 revision. (page 181)

[124] B .T. Polyak and Y. Z. Tsypkin. Pseudogradient adaptation and training algorithms. *Automation and Remote Control*, 34:45–67, 1973. (pages 19, 61, and 83)

[125] Z. Qu, P. Richtárik, and T. Zhang. Quartz: Randomized dual coordinate ascent with arbitrary sampling. In *Proceedings of the 29th Annual Conference on Neural Information Processing Systems (NIPS)*. 2015. (page 223)

[126] A. Ramdas, S. J. Reddi, B. Póczos, A. Singh, and L. A. Wasserman. Adaptivity and computation-statistics tradeoffs for kernel and distance based high dimensional two sample testing. *CoRR*, abs/1508.00655, 2015. (pages 13 and 249)

[127] L. A. Rastrigin. *Statisticheskie Metody Poiska Ekstremuma (Statistical Extremum Seeking Methods)*. Nauka, Moscow, 1968. (page 181)

[128] B. Recht, C. Re, S. J. Wright, and F. Niu. Hogwild!: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent. In *Proceedings of the 25th Annual Conference on Neural Information Processing Systems (NIPS)*, pages 693–701, 2011. (pages 10, 46, 149, 150, 154, 155, 158, 159, 160, 182, 185, 193, 194, and 222)

[129] S. J. Reddi and E. Brunskill. Incentive decision processes. In *Proceedings of the Twenty-Eighth Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 418–427, 2012. (page 13)

[130] S. J. Reddi and B. Póczos. Scale invariant conditional dependence measures. In *Proceedings of the 30th International Conference on Machine Learning (ICML)*, pages 1355–1363, 2013. (pages 13 and 249)

[131] S. J. Reddi and B. Póczos. k-nn regression on functional data with incomplete observations. In *Proceedings of the Thirtieth Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 692–701, 2014. (pages 13 and 249)

[132] S. J. Reddi, A. Hefny, C. Downey, A. Dubey, and S. Sra. Large-scale randomized-coordinate descent methods with non-separable linear constraints. In *Proceedings of the Thirty-First Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 752–761, 2015. (page 13)

[133] S. J. Reddi, A. Hefny, S. Sra, B. Poczos, and A. Smola. On variance reduction in stochastic gradient descent and its asynchronous variants. In *Proceedings of the 29th Annual Conference on Neural Information Processing Systems (NIPS)*, pages 2629–2637, 2015. (pages 13, 18, 46, 61, 83, 87, and 222)

[134] S. J. Reddi, B. Póczos, and A. J. Smola. Communication efficient coresets for empirical risk minimization. In *Proceedings of the Thirty-First Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 762–771, 2015. (page 13)

[135] S. J. Reddi, B. Póczos, and A. J. Smola. Doubly robust covariate shift correction. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, pages 2949–2955, 2015. (page 13)

[136] S. J. Reddi, A. Ramdas, B. Póczos, A. Singh, and L. A. Wasserman. On the decreasing power of kernel and distance based nonparametric hypothesis tests in high dimensions. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial*

Intelligence, pages 3571–3577, 2015. (pages 13 and 249)

[137] S. J. Reddi, A. Ramdas, B. Póczos, A. Singh, and L. A. Wasserman. On the high dimensional power of a linear-time two sample test under mean-shift alternatives. In *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2015. (pages 13 and 249)

[138] S. J. Reddi, A. Dubey, B. Póczos, A. J. Smola, E. Xing, and S. Williamson. Variance reduction in stochastic gradient Langevin dynamics. In *Proceedings of the 30th Annual Conference on Neural Information Processing Systems (NIPS)*, 2016. (page 13)

[139] S. J. Reddi, A. Hefny, S. Sra, B. Póczos, and A. J. Smola. Stochastic variance reduction for nonconvex optimization. In *Proceedings of the 33nd International Conference on Machine Learning, ICML*, pages 314–323, 2016. (pages 13, 61, 64, 65, 72, 76, 83, 84, 89, 105, 110, 111, 124, and 125)

[140] S. J. Reddi, J. Konecný, P. Richtárik, B. Póczos, and A. J. Smola. AIDE: fast and communication efficient distributed optimization. *CoRR*, abs/1608.06879, 2016. (page 13)

[141] S. J. Reddi, S. Sra, B. Póczos, and A. J. Smola. Fast incremental method for nonconvex optimization. In *Proceedings of the 55th IEEE Conference on Decision and Control (CDC)*, 2016. (pages 13, 61, 83, 84, 89, 110, 111, 119, and 124)

[142] S. J. Reddi, S. Sra, B. Póczos, and A. J. Smola. Proximal stochastic methods for nonsmooth nonconvex finite-sum optimization. In *Proceedings of the 30th Annual Conference on Neural Information Processing Systems (NIPS)*, 2016. (pages 13, 61, and 111)

[143] S. J. Reddi, S. Sra, B. Póczos, and A. J. Smola. Stochastic frank-wolfe methods for nonconvex optimization. In *54th Annual Allerton Conference on Communication, Control, and Computing*, 2016. (page 13)

[144] P. Richtárik and M. Takáč. Iteration complexity of randomized block-coordinate descent methods for minimizing a composite function. *Mathematical Programming*, 144(1-2):1–38, 2014. (page 150)

[145] P. Richtárik and M. Takáč. Distributed coordinate descent method for learning with big data. *CoRR*, abs/1310.2059, 2013. (pages 181, 182, and 190)

[146] P. Richtárik and M. Takáč. Iteration complexity of randomized block-coordinate descent methods for minimizing a composite function. *CoRR*, abs/1107.2848, 2011. (pages 181, 182, 183, 184, and 190)

[147] P. Richtárik and M. Takáč. Parallel coordinate descent methods for big data optimization. *CoRR*, abs/1212.0873, 2012. (pages 182, 183, and 190)

[148] H. Robbins and S. Monro. A stochastic approximation method. *Annals of Mathematical Statistics*, 22:400–407, 1951. (pages 6, 17, 19, 46, 61, 83, and 149)

[149] H. Robbins and S. Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, 22(3):400–407, sep 1951. (pages 10, 113, 129, and 130)

[150] R. T. Rockafellar. Monotone operators and the proximal point algorithm. *SIAM*

Journal on Control and Optimization, 14(5):877–898, 1976. (page 81)

[151] A. Saha and A. Tewari. On the nonasymptotic convergence of cyclic coordinate descent methods. *SIAM Journal on Optimization*, 23(1):576–601, 2013. (page 181)

[152] K. Scheinberg. An efficient implementation of an active set method for svms. *Journal of Machine Learning Research*, 7:2237–2257, December 2006. ISSN 1532-4435. (page 207)

[153] M. W. Schmidt, N. L. Roux, and F. R. Bach. Minimizing Finite Sums with the Stochastic Average Gradient. *Mathematical Programming*. (pages 17, 18, 29, 46, 50, 61, 83, 92, 110, 130, 132, 136, 149, 150, 151, 158, 207, and 229)

[154] S. Shalev-Shwartz. SDCA without duality. *CoRR*, abs/1502.06177, 2015. (pages 18, 24, 46, 83, 91, 230, and 243)

[155] S. Shalev-Shwartz and T. Zhang. Accelerated mini-batch stochastic dual coordinate ascent. In *Proceedings of the 27th Annual Conference on Neural Information Processing Systems (NIPS)*, pages 378–385, 2013. (page 150)

[156] S. Shalev-Shwartz and T. Zhang. Stochastic dual coordinate ascent methods for regularized loss. *The Journal of Machine Learning Research*, 14(1):567–599, 2013. (pages 18, 46, 61, 83, 149, 150, and 229)

[157] S. Shalev-Shwartz and T. Zhang. Stochastic dual coordinate ascent methods for regularized loss minimization. *Journal of Machine Learning Research (JMLR)*, 14, 2013. (pages 181 and 182)

[158] S. Shalev-Shwartz and T. Zhang. Accelerated mini-batch stochastic dual coordinate ascent. *CoRR*, abs/1305.2581, 2013. (page 182)

[159] S. Shalev-Shwartz, Y. Singer, and N. Srebro. Pegasos: Primal estimated subgradient solver for svm. In *Proceedings of the 24th International Conference on Machine Learning (ICML)*, pages 807–814, 2007. (page 207)

[160] O. Shamir. A stochastic PCA and SVD algorithm with an exponential convergence rate. *CoRR*, abs/1409.2848, 2014. (pages 19, 46, and 84)

[161] O. Shamir. Fast stochastic algorithms for SVD and PCA: Convergence properties and convexity. *CoRR*, abs/1507.08788, 2015. (pages 19 and 84)

[162] O. Shamir and N. Srebro. On distributed stochastic optimization and learning. In *Proceedings of the 52nd Annual Allerton Conference on Communication, Control, and Computing*, 2014. (page 149)

[163] O. Shamir, N. Srebro, and T. Zhang. Communication efficient distributed optimization using an approximate newton-type method. *CoRR*, abs/1312.7853, 2013. (pages 12, 222, 223, 225, 226, 233, 234, 238, 239, 240, and 241)

[164] S. Sra. Scalable nonconvex inexact proximal splitting. In *Proceedings of the 26th Annual Conference on Neural Information Processing Systems (NIPS)*, pages 530–538, 2012. (pages 19, 46, and 84)

[165] I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initial-

ization and momentum in deep learning. In *Proceedings of the 30th International Conference on Machine Learning (ICML)*, pages 1139–1147, 2013. (page 69)

[166] R. Tappenden, P. Richtárik, and J. Gondzio. Inexact coordinate descent: complexity and preconditioning. *CoRR*, abs/1304.5530, 2013. (page 182)

[167] R. Tibshirani, M. Saunders, S. Rosset, J. Zhu, and K. Knight. Sparsity and smoothness via the fused lasso. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(1):91–108, 2005. (page 182)

[168] I. W. Tsang, J. T. Kwok, P. Cheung, and N. Cristianini. Core vector machines: Fast svm training on very large data sets. *Journal of Machine Learning Research*, 6: 363–392, 2005. (page 207)

[169] P. Tseng and S. Yun. A block-coordinate gradient descent method for linearly constrained nonsmooth separable optimization. *Journal of Optimization Theory and Applications*, 2009. (pages 182 and 184)

[170] O. Vinyals and D. Povey. Krylov subspace descent for deep learning. In *Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 1261–1268, 2012. (page 69)

[171] P. Wang and C. Lin. Iteration complexity of feasible descent methods for convex optimization. *Journal of Machine Learning Research (JMLR)*, 15:1523–1548, 2014. (pages 181 and 183)

[172] M. Welling and Y. W. Teh. Bayesian learning via stochastic gradient Langevin dynamics. In *Proceedings of the 28th International Conference on Machine Learning (ICML)*, 2011. (pages 10, 129, 130, and 136)

[173] L. Xiao and T. Zhang. A proximal stochastic gradient method with progressive variance reduction. *SIAM Journal on Optimization*, 24(4):2057–2075, 2014. (pages 21, 25, 81, 83, 85, 111, 149, 150, and 159)

[174] L. Xiao and T. Zhang. A proximal stochastic gradient method with progressive variance reduction. *SIAM Journal on Optimization*, 24(4):2057–2075, 2014. (page 219)

[175] Yossi Y. Arjevani and O. Shamir. Communication complexity of distributed convex learning and optimization. In *Proceedings of the 29th Annual Conference on Neural Information Processing Systems (NIPS)*, pages 1747–1755. 2015. (pages 12, 223, 224, 225, 227, 228, and 231)

[176] A. Kyrola D. Bickson C. Guestrin Y. Low, J. Gonzalez and J. M. Hellerstein. Distributed GraphLab: A Framework for Machine Learning and Data Mining in the Cloud. *PVLDB*, 2012. (pages 2 and 221)

[177] T. Yang. Trading computation for communication: Distributed stochastic dual coordinate ascent. In *Proceedings of the 27th Annual Conference on Neural Information Processing Systems (NIPS)*, pages 629–637, 2013. (page 222)

[178] Y. Yu, X. Zhang, and D. Schuurmans. Generalized conditional gradient for sparse estimation. abs/arXiv:1410.4828, 2014. (page 110)

[179] A. Z. Zeyuan and Y. Yuan. Improved svrg for non-strongly-convex or sum-of-non-convex objectives. *CoRR*, abs/1506.01972, 2015. (pages 25, 83, and 91)

[180] Y. Zhang and X. Lin. Disco: Distributed optimization for self-concordant empirical loss. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, pages 362–370, 2015. (pages 12 and 222)

[181] Y. Zhang, M. J. Wainwright, and J. C. Duchi. Communication-efficient algorithms for statistical optimization. In *Proceedings of the 26th Annual Conference on Neural Information Processing Systems (NIPS)*, pages 1502–1510, 2012. (page 223)

[182] M. Zinkevich, M. Weimer, L. Lihong, and A.J. Smola. Parallelized stochastic gradient descent. In *Proceedings of the 24th Annual Conference on Neural Information Processing Systems (NIPS)*, pages 2595–2603, 2010. (pages 149, 207, and 223)