2019

Distributed Algorithms in Large-scaled Empirical Risk Minimization: Non-convexity, Adaptive-sampling, and Matrix-free Second-order Methods

Xi He
Lehigh University, heeryerate@gmail.com

Distributed Algorithms in Large-scaled Empirical Risk Minimization: Non-convexity, Adaptive Sampling, and Matrix-free Second-order Methods

by

Xi He

Presented to the Graduate and Research Committee

of Lehigh University

in Candidacy for the Degree of

Doctor of Philosophy

in

Industrial and Systems Engineering

Lehigh University

January 2019

Approved and recommended for acceptance as a dissertation in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

_____
Date

_____
Dr. Martin Takáč, Dissertation Advisor

Committee Members:

_____
Dr. Martin Takáč, Committee Chair

_____
Dr. Katya Scheinberg

_____
Dr. Frank E. Curtis

_____
Dr. Martin Jaggi

Acknowledgments

Firstly, I would like to express my sincere gratitude to my advisor Dr. Martin Takáč for his continuous support of my overall Ph.D. process, for his patience, motivation, immense knowledge and brainstorm ideas. I would never forget the very first time when he helped me accelerate my naive code to let it run ten times faster, like a magic. Dr. Martin Takáč is not only an advisor for me, but a mentor for many aspects in my life. All the relevant research and this dissertation are certainly impossible to complete without the constant and generous help from him.

Besides my advisor, I would also thank all my dissertation committee members: Dr. Katya Scheinberg, Dr. Frank E. Curtis, and Dr. Martin Jaggi, not only for your insightful comments and encouragement, but the hard questions which help me to understand and widen my research in various perspectives. My deep thanks would also go to our Optimization and Machine Learning (OptML) group at Lehigh University. I always feel extremely lucky to be a member of OptML, where I'm able to exchange ideas and discuss interests with all members. Without the year by year's weekly seminar and insightful discussion, I would be definitely not able to build my addictive interests in the field of optimization.

I'm also grateful to Rachael Tappenden, Albert Berahas, and Dheevatsa Mudiger. With all their endless help, I would be able to stay on the right track towards my interests. Also, I would like to especially thank Ioannis Akrotirianakis, Amit Chakraborty, Neo Hsin-Chan Huang, Shuo Li, Eric Zhu, Maxence G Hardy, who provided me opportunities to join their team as an intern. With more touch of real-world applications matching what I have learned, I would, therefore, be clear on my future direction.

All my friends at Lehigh University deserve special thanks, Chenxin Ma, Yuhai Hu, Wei Xia,

Yinan Liu, Rui Shi, Miao Bai, Choat Inthawongse, Mohammadreza Samadi, Hiva Ghanbari, and many others. I can't stop to recall all the fun we have had in the last several years and all their support and kindness, they make my life at Bethlehem so wonderful. I would also borrow the place to thank my previous advisor, Qingzhi Yang and all my friends back at Nankai University, China. Without your encouragement and understanding, any of the overseas life would not happen.

Finally, I am deeply indebted to my family for their tremendous love and consistently supporting of all my decisions. They have been always with me throughout my whole Ph.D. period. Wishing you all my family and friends good health and happiness forever.

Contents

List of Tables

List of Figures

Abstract

The rising amount of data has changed the classical approaches in statistical modeling significantly. Special methods are designed for inferring meaningful relationships and hidden patterns from these large datasets, which build the foundation of a study called Machine Learning (ML). Such ML techniques have already applied widely in various areas and achieved compelling success.

In the meantime, the huge amount of data also requires a deep revolution of current techniques, like the availability of advanced data storage, new efficient large-scale algorithms and their distributed/parallelized implementation.

There is a broad class of ML methods can be interpreted as Empirical Risk Minimization (ERM) problems. When utilize various loss functions and likely necessary regularization terms, one could approach their specific ML goals by solving ERMs as separable finite sum optimization problems. There are circumstances where nonconvex component is introduced into the ERMs which usually makes the problems hard to optimize. Especially, in recent years, neural networks, a popular branch of ML, draw numerous attention from community. Neural networks are powerful and highly flexible inspired by the structured functionality of the brain. Typically, neural networks could be treated as large-scale and highly nonconvex ERMs.

While as nonconvex ERMs become more complex and larger in scales, optimization using stochastic gradient descent (SGD) type methods proceeds slowly regarding its convergence rate and incapability of being distributed efficiently. It motivates researchers to explore more advanced local optimization methods such as approximate-Newton/second-order methods.

In this dissertation, first-order stochastic optimization for the regularized ERMs in Chapter1

is studied. Based on the development of stochastic dual coordinate accent (SDCA) method, a dual free SDCA with non-uniform mini-batch sampling strategy is investigated [30, 29]. We also introduce several efficient algorithms for training ERMs, including neural networks, using second-order optimization methods in a distributed environment. In Chapter 2, we propose a practical distributed implementation for Newton-CG methods. It makes training neural networks by second-order methods doable in the distributed environment [28]. In Chapter 3, we further build steps towards using second-order methods to train feed-forward neural networks with negative curvature direction utilization and momentum acceleration. In this Chapter, we also report numerical experiments for comparing second-order methods and first-order methods regarding training neural networks . The following Chapter 4 purpose an distributed accumulative sample-size second-order methods for solving large scale convex ERMs and nonconvex neural networks [35]. In Chapter 5, a python library named UCLibrary is briefly introduced for solving unconstrained optimization problems. This dissertation is all concluded in the last Chapter 6.

Chapter 1

Dual Free Adaptive Mini-batch SDCA for Empirical Risk Minimization

1.1 Introduction

In this chapter we study the ℓ_2-regularized Empirical Risk Minimization (ERM) problem, which is widely used in the field of machine learning. The problem can be stated as follows. Given training examples $(x_1, y_1), \ldots, (x_n, y_n) \in \mathbb{R}^d \times \mathbb{R}$, loss functions $\phi_1, \ldots, \phi_n : \mathbb{R} \to \mathbb{R}$ and a regularization parameter $\lambda > 0$, ℓ_2-regularized ERM is an optimization problem of the form

$$\min_{w \in \mathbb{R}^d} P(w) := \frac{1}{n} \sum_{i=1}^{n} \phi_i(w^T x_i) + \frac{\lambda}{2} \|w\|^2, \tag{1.1}$$

where the first term in the objective function is a *data fitting term* and the second is a *regularization term* that prevents over-fitting.

Many algorithms have been proposed to solve problem (1.1) over the past few years, including SGD, [84], SVRG and S2GD, [36, 64, 40] and SAG/SAGA, [79, 18, 76]. However, another very popular approach to solving ℓ_2-regularized ERM problems is to consider the following dual formulation

$$\max_{\alpha \in \mathbb{R}^n} D(\alpha) := -\frac{1}{n}\sum_{i=1}^{n} \phi_i^*(-\alpha_i) - \frac{\lambda}{2}\|\frac{1}{\lambda n} X^T \alpha\|^2, \tag{1.2}$$

where $X^T = [x_1, \ldots, x_n] \in \mathbb{R}^{d \times n}$ is the data matrix and ϕ_i^* denotes the Fenchel conjugate of ϕ_i, namely, $\phi_i^*(u) = \max_z(zu - \phi_i(z))$. It is also known that $P(w^*) = D(\alpha^*)$, which implies that for all w and α, we have $P(w) \geq D(\alpha)$, and hence the duality gap, defined to be $P(w(\alpha)) - D(\alpha)$, can be regarded as an upper bound on the primal sub-optimality $P(w(\alpha)) - P(w^*)$. The structure of the dual formulation (1.2) makes it well suited to a multi-core or distributed computational setting, and several algorithms have been developed to take advantage of this including [32] [93, 34, 49, 94, 73, 13, 104].

A popular method for solving (1.2) is Stochastic Dual Coordinate Ascent (SDCA). The algorithm proceeds as follows. At iteration t of SDCA a coordinate $i \in \{1, \ldots, n\}$ is chosen uniformly at random and the current iterate $\alpha^{(t)}$ is updated to $\alpha^{(t+1)} := \alpha^{(t)} + \delta^* e_i$, where $\delta^* = \arg\max_{\delta \in \mathbb{R}} D(\alpha^{(t)} + \delta e_i)$. Much research has focused on analysing the theoretical complexity of SDCA under various assumptions imposed on the functions ϕ_i^*, including the pioneering work of Nesterov in [60] and others including [75, 95, 58, 57, 47, 94, 93].

A modification that has led to improvements in the practical performance of SDCA is the use of *importance sampling* when selecting the coordinate to update. That is, rather than using uniform probabilities, instead coordinate i is sampled with an arbitrary probability p_i, see for example [105, 13].

In many cases algorithms that employ non-uniform coordinate sampling outperform naïve uniform selection, and in some cases help to decrease the number of iterations needed to achieve a desired accuracy by several orders of magnitude, see for example [105, 13].

Notation and Assumptions. In this chapter we use the notation $[n] \stackrel{\text{def}}{=} \{1, \ldots, n\}$, as well as the following assumption. For all $i \in [n]$, the loss function ϕ_i is $\tilde{L}_i$-smooth with $\tilde{L}_i > 0$, i.e., for any given $\beta, \delta \in \mathbb{R}$, we have

$$|\phi_i'(\beta) - \phi_i'(\beta + \delta)| \leq \tilde{L}_i |\delta|. \tag{1.3}$$

In addition, it is simple to observe that the function $\phi_i(x_i^T \cdot) : \mathbb{R}^d \to \mathbb{R}$ is L_i smooth, i.e.,

$\forall w, \bar{w} \in \mathbb{R}^d$ and for all $i \in [n]$ there exists a constant $L_i \leq \|x_i\|^2 \tilde{L}_i$ such that

$$\|\nabla \phi_i(x_i^T w) - \nabla \phi_i(x_i^T \bar{w})\| \leq L_i \|w - \bar{w}\|. \tag{1.4}$$

We will use the notation

$$L = \max_{1 \leq i \leq n} L_i, \qquad \text{and} \qquad \tilde{L} = \max_{1 \leq i \leq n} \tilde{L}_i. \tag{1.5}$$

Throughout this chapter we let $\mathbb{R}_+$ denote the set of nonnegative real numbers and we let $\mathbb{R}_+^n$ denote the set of n-dimensional vectors with all components being real and nonnegative.

1.1.1 Contributions

In this section the main contributions of this chapter are summarized (not in order of significance).

Adaptive SDCA. We modify the dual free SDCA algorithm proposed in [83] to allow for the adaptive adjustment of probabilities and a non-uniform selection of coordinates. Note that the method is dual free, and hence in contrast to classical SDCA, where the update is defined by maximizing the dual objective (1.2), here we define the update slightly differently (see Section 1.2 for details).

Allowing non-uniform selection of coordinates from an adaptive probability distribution leads to improvements in practical performance and the algorithm achieves a better convergence rate than in [83]. In short, we show that the error after T iterations is decreased by a factor of $\prod_{t=1}^{T}(1 - \theta^{(t)}) \geq (1 - \theta^*)^T$ on average, where θ^* is an uniformly lower bound for all $\theta^{(t)}$. Here $1 - \theta^{(t)} \in (0, 1)$ is a parameter that depends on the current iterate $\alpha^{(t)}$ and the nonuniform probability distribution. By changing the coordinate selection strategy from uniform selection to adaptive, each $1 - \theta^{(t)}$ becomes smaller, which leads to an improvement in the convergence rate.

Non-uniform sampling procedure. Rather than using a uniform sampling of coordinates, which is the commonly used approach, here we propose the use of non-uniform sampling from an adaptive probability distribution. With this novel sampling strategy, we

are able to generate non-uniform non-overlapping and proper (see Section 1.5) samplings for arbitrary marginal distributions under only one mild assumptions. Indeed, we show that without the assumption, there is no such non-uniform sampling strategy. We also extend our sampling strategy to allow the selection of mini-batches.

Better convergence and complexity results. By utilizing an adaptive probabilities strategy, we can derive complexity results for our new algorithm that, for the case when every loss function is convex, depend only on the *average* of the Lipschitz constants L_i. This improves upon the complexity theory developed in [83] (which uses a uniform sampling) and [14] (which uses an arbitrary but fixed probability distribution), because the results in those works depend on the *maximum* Lipschitz constant. Furthermore, even though adaptive probabilities are used here, we are still able to retain the very nice feature of the work in [83], and show that the variance of the update naturally goes to zero as the iterates converge to the optimum without any additional computational effort or storage costs. Our adaptive probabilities SDCA method also comes with an improved bound on the variance of the update in terms of the sub-optimality of the current iterate.

Practical aggressive variant. Following from the work of [13], we propose an efficient heuristic variant of adfSDCA. For adfSDCA the adaptive probabilities must be computed at every iteration (i.e., once a single coordinate has been selected), which can be computationally expensive. However, for our heuristic adfSDCA variant the (exact/true) adaptive probabilities are only computed once at the beginning of each epoch (where an epoch is one pass over the data/n coordinate updates), and during that epoch, once a coordinate has been selected we simply reduce the probability associated with that coordinate so it is not selected again during that epoch. Intuitively this is reasonable because, after a coordinate has been updated the dual residue associated with that coordinate decreases and thus the probability of choosing this coordinate should also reduce. We show that in practice this heuristic adfSDCA variant converges and the computational effort required by this algorithm is lower than adfSDCA (see Sections 1.4 and 1.6).

Mini-batch variant. We extend the (serial) adfSDCA algorithm to incorporate a mini-batch scheme. The motivation for this approach is that there is a computational cost associated with generating the adaptive probabilities, so it is important to utilize them

effectively. We develop a non-uniform mini-batch strategy that allows us to update multiple coordinates in one iteration, and the coordinates that are selected have high potential to decrease the sub-optimality of the current iterate. Further, we make use of ESO framework (Expected Separable Overapproximation) (see for example [74], [73]) and present theoretical complexity results for mini-batch adfSDCA. In particular, for mini-batch adfSDCA used with batchsize b, we derive the optimal probabilities to use at each iteration, as well as the best step-size to use to guarantee speedup.

1.1.2 Outline

This chapter is organized as follows. In Section 1.2 we introduce our new Adaptive Dual Free SDCA algorithm (adfSDCA), and highlight its connection with a reduced variance SGD method. In Section 1.3 we provide theoretical convergence guarantees for adfSDCA in the case when all loss functions $\phi_i(\cdot)$ are convex, and also in the case when individual loss functions are allowed to be nonconvex but the average loss functions $\sum_{i=1}^n \phi_i(\cdot)$ is convex. Section 1.4 introduces a practical heuristic version of adfSDCA, and in Section 1.5 we present a mini-batch adfSDCA algorithm and provide convergence guarantees for that method. Finally, we present the results of our numerical experiments in Section 1.6. Note that the proofs for all the theoretical results developed in this chapter are left to the appendix.

1.2 The Adaptive Dual Free SDCA Algorithm

In this section we describe the Adaptive Dual Free SDCA (adfSDCA) algorithm, which is motivated by the dual free SDCA algorithm proposed by [83]. Note that in dual free SDCA two sequences of primal and dual iterates, $\{w^{(t)}\}_{t=0}^{\infty}$ and $\{\alpha^{(t)}\}_{t=0}^{\infty}$ respectively, are maintained. At every iteration of that algorithm, the variable updates are computed in such a way that the well known primal-dual relational mapping holds; for every iteration t:

$$w^{(t)} = \frac{1}{\lambda n}\sum_{i=1}^{n} \alpha_i^{(t)} x_i. \tag{1.6}$$

The dual residue is defined as follows.

Definition 1.2.1 (Dual residue, [13]). *The dual residue* $\kappa^{(t)} = (\kappa_1^{(t)}, \ldots, \kappa_n^{(t)})^T \in \mathbb{R}^n$ *associated with* $(w^{(t)}, \alpha^{(t)})$ *is given by:*

$$\kappa_i^{(t)} \stackrel{def}{=} \alpha_i^{(t)} + \phi_i'(x_i^T w^{(t)}). \tag{1.7}$$

The Adaptive Dual Free SDCA algorithm is outlined in Algorithm 1.1 and is described briefly now; a more detailed description (including a discussion of coordinate selection and how to generate appropriate selection rules) will follow. An initial solution $\alpha^{(0)}$ is chosen, and then $w^{(0)}$ is defined via (1.6). In each iteration of Algorithm 1.1 the dual residue $\kappa^{(t)}$ is computed via (1.7), and this is used to generate a probability distribution $p^{(t)}$. Next, a coordinate $i \in [n]$ is selected (sampled) according to the generated probability distribution and a step of size $\theta^{(t)} \in (0, 1)$ is taken by updating the ith coordinate of α via

$$\alpha_i^{(t+1)} = \alpha_i^{(t)} - \theta^{(t)}(p_i^{(t)})^{-1}\kappa_i^{(t)}. \tag{1.8}$$

Finally, the vector w is also updated

$$w^{(t+1)} = w^{(t)} - \theta^{(t)}(n\lambda p_i^{(t)})^{-1}\kappa_i^{(t)} x_i, \tag{1.9}$$

and the process is repeated. Note that the updates to α and w using the formulas (1.8) and (1.9) ensure that the equality (1.6) is preserved.

Also note that the updates in (1.8) and (1.9) involve a step size parameter $\theta^{(t)}$, which will play an important role in our complexity results. The step size $\theta^{(t)}$ should be large so that good progress can be made, but it must also be small enough to ensure that the algorithm is guaranteed to converge. Indeed, in Section 1.3.1 we will see that the choice of $\theta^{(t)}$ depends on the choice of probabilities used at iteration t, which in turn depend upon a particular function that is related to the suboptimality at iteration t.

The dual residue $\kappa^{(t)}$ is informative and provides a useful way of monitoring suboptimality of the current solution $(w^{(t)}, \alpha^{(t)})$. In particular, note that if $\kappa_i = 0$ for some coordinate i, then by (1.7) $\alpha_i = -\phi_i'(w^T x_i)$, and substituting κ_i into (1.8) and (1.9) shows that $\alpha_i^{(t+1)} \leftarrow \alpha_i^{(t)}$ and $w_i^{(t+1)} \leftarrow w^{(t)}$, i.e., α and w remain unchanged in that iteration.

Algorithm 1.1 Adaptive Dual Free SDCA (adfSDCA)

Input: Data: $\{x_i, \phi_i\}_{i=1}^n$
Initialization: Choose $\alpha^{(0)} \in \mathbb{R}^n$
Set $w^{(0)} = \frac{1}{\lambda n} \sum_{i=1}^n \alpha_i^{(0)} x_i$
for $t = 0, 1, 2, \ldots$ **do**
  Calculate dual residual $\kappa_i^{(t)} = \phi_i'(x_i^T w^{(t)}) + \alpha_i^{(t)}$, for all $i \in [n]$
  Generate adaptive probability distribution $p^{(t)} \sim \kappa^{(t)}$
  Sample coordinate i according to $p^{(t)}$
  Set step-size $\theta^{(t)} \in (0, 1)$ as in (1.20)
  Update: $\alpha_i^{(t+1)} = \alpha_i^{(t)} - \theta^{(t)} (p_i^{(t)})^{-1} \kappa_i^{(t)}$
  Update: $w^{(t+1)} = w^{(t)} - \theta^{(t)} (n\lambda p_i^{(t)})^{-1} \kappa_i^{(t)} x_i$
end for

On the other hand, a large value of $|\kappa_i|$ (at some iteration t) indicates that a large step will be taken, which is anticipated to lead to good progress in terms of improvement in sub-optimality of current solution.

The probability distributions used in Algorithm 1.1 adhere to the following definition.

Definition 1.2.2. *(Coherence, [13]) Probability vector $p \in \mathbb{R}^n$ is coherent with dual residue $\kappa \in \mathbb{R}^n$ if for any index i in the support set of κ, denoted by $I_\kappa := \{i \in [n] : \kappa_i \neq 0\}$, we have $p_i > 0$. When $i \notin I_\kappa$ then $p_i = 0$. We use $p \sim \kappa$ to represent this coherent relation.*

1.2.1 Adaptive dual free SDCA as a reduced variance SGD method.

Reduced variance SGD methods have became very popular in the past few years, see for example [41, 36, 76, 18]. It is show in [83] that uniform dual free SDCA is an instance of a reduced variance SGD algorithm (the variance of the stochastic gradient can be bounded by some measure of sub-optimality of the current iterate) and a similar result applies to adfSDCA in Algorithm 1.1. In particular, note that conditioned on $\alpha^{(t-1)}$, we have

$$\begin{aligned}
\mathbb{E}[w^{(t)} | \alpha^{(t-1)}] &\overset{(1.9)}{=} w^{(t-1)} - \frac{\theta^{(t-1)}}{\lambda} \sum_{i=1}^n \frac{p_i}{np_i} \Big(\big(\nabla \phi_i(x_i^T w^{(t-1)}) + \alpha_i^{(t-1)} \big) x_i \Big) \\
&\overset{(1.6)}{=} w^{(t-1)} - \frac{\theta^{(t-1)}}{\lambda} \Big(\nabla \Big(\frac{1}{n} \sum_{i=1}^n \phi_i(x_i^T w^{(t-1)}) \Big) + \lambda w^{(t-1)} \Big) \\
&\overset{(1.1)}{=} w^{(t-1)} - \frac{\theta^{(t-1)}}{\lambda} \nabla P(w^{(t-1)}). \qquad (1.10)
\end{aligned}$$

Combining (1.9) and (1.10) and replace $t-1$ by t gives

$$\mathbb{E}\left[\frac{1}{np_i}\kappa_i^{(t)}x_i|\alpha^{(t)}\right] = \nabla P(w^{(t)}), \tag{1.11}$$

which implies that $\frac{1}{np_i}\kappa_i^{(t)}x_i$ is an unbiased estimator of $\nabla P(w^{(t)})$. Therefore, Algorithm 1.1 is eventually a variant of the Stochastic Gradient Descent method. However, we can prove (see Corollary 1.3.4 and Corollary 1.3.7) that the variance of the update goes to zero as the iterates converge to an optimum, which is not true for vanilla Stochastic Gradient Descent.

1.3 Convergence Analysis

In this section we state the main convergence results for adfSDCA (Algorithm 1.1). The analysis is broken into two cases. In the first case it is assumed that each of the loss functions ϕ_i is convex. In the second case this assumption is relaxed slightly and it is only assumed that the average of the ϕ_i's is convex, i.e., individual functions $\phi_i(\cdot)$ for some (several) $i \in [n]$ are allowed to be nonconvex, as long as $\frac{1}{n}\sum_{j=1}^{n}\phi_j(\cdot)$ is convex. The proofs for all the results in this section can be found in the Appendix.

1.3.1 Case I: All loss functions are convex

Here we assume that ϕ_i is convex for all $i \in [n]$. Define the following parameter

$$\gamma \stackrel{\text{def}}{=} \lambda\tilde{L}, \tag{1.12}$$

where $\tilde{L}$ is given in (1.5). It will also be convenient to define the following potential function. For all iterations $t \geq 0$,

$$D^{(t)} \stackrel{\text{def}}{=} \tfrac{1}{n}\|\alpha^{(t)} - \alpha^*\|^2 + \gamma\|w^{(t)} - w^*\|^2. \tag{1.13}$$

The potential function (1.13) plays a central role in the convergence theory presented in this chapter. It measures the distance from the optimum in both the primal and (pseudo) dual variables. Thus, our algorithm will generate iterates that reduce this suboptimality

and therefore push the potential function toward zero.

Also define

$$v_i \stackrel{\text{def}}{=} \|x_i\|^2 \text{ for all } i \in [n]. \tag{1.14}$$

We have the following result.

Lemma 1.3.1. *Let $\tilde{L}$, $\kappa_i^{(t)}$, γ, $D^{(t)}$, and v_i be as defined in* (1.5), (1.7), (1.12), (1.13) *and* (1.14)*, respectively. Suppose that ϕ_i is $\tilde{L}$-smooth and convex for all $i \in [n]$ and let $\theta \in (0,1)$. Then at every iteration $t \geq 0$ of Algorithm 1.1, a probability distribution $p^{(t)}$ that satisfies Definition 1.2.2 is generated and*

$$\mathbb{E}\big[D^{(t+1)}|\alpha^{(t)}\big] - (1-\theta)D^{(t)} \leq \sum_{i=1}^{n}\left(-\frac{\theta}{n}\left(1 - \frac{\theta}{p_i^{(t)}}\right) + \frac{\theta^2 v_i \gamma}{n^2\lambda^2 p_i^{(t)}}\right)(\kappa_i^{(t)})^2. \tag{1.15}$$

Note that if the right hand side of (1.15) is negative, then the potential function decreases (in expectation) in iteration t:

$$\mathbb{E}\big[D^{(t+1)}|\alpha^{(t)}\big] \leq (1-\theta)D^{(t)}. \tag{1.16}$$

The purpose of Algorithm 1.1 is to generate iterates $(w^{(t)}, \alpha^{(t)})$ such that the above holds. To guarantee negativity of the right hand term in (1.15), or equivalently, to ensure that (1.16) holds, consider the parameter θ. Specifically, any θ that is less than the function $\Theta(\cdot,\cdot) : \mathbb{R}_+^n \times \mathbb{R}_+^n \to \mathbb{R}$ defined as

$$\Theta(\kappa, p) \stackrel{\text{def}}{=} \frac{n\lambda^2 \sum_{i\in I_\kappa} \kappa_i^2}{\sum_{i\in I_\kappa}(n\lambda^2 + v_i\gamma)p_i^{-1}\kappa_i^2}, \tag{1.17}$$

will ensure negativity of the right hand term in (1.15). Moreover, the larger the value of θ, the better progress Algorithm 1.1 will make in terms of the reduction in $D^{(t)}$. The function Θ depends on the dual residue κ and the probability distribution p. Maximizing this function w.r.t. p will ensure that the largest possible value of θ can be used in Algorithm 1.1. Thus, we consider the following optimization problem:

$$\max_{p\in\mathbb{R}_+^n, \sum_{i\in I_\kappa} p_i = 1} \quad \Theta(\kappa, p). \tag{1.18}$$

One may naturally be wary of the additional computational cost incurred by solving the optimization problem in (1.18) at every iteration. Fortunately, it turns out that there is an (inexpensive) closed form solution, as shown by the following Lemma.

Lemma 1.3.2. *Let* $\Theta(\kappa, p)$ *be defined in* (1.17). *The optimal solution* $p^*(\kappa)$ *of* (1.18) *is*

$$p_i^*(\kappa) = \frac{\sqrt{v_i\gamma + n\lambda^2}|\kappa_i|}{\sum_{j\in I_\kappa}\sqrt{v_j\gamma + n\lambda^2}|\kappa_j|}, \qquad \text{for all } \; i = 1, \ldots, n. \tag{1.19}$$

The corresponding θ *by using the optimal solution* p^* *is*

$$\theta = \Theta(\kappa, p^*) = \frac{n\lambda^2 \sum_{i\in I_\kappa}\kappa_i^2}{(\sum_{i\in I_\kappa}\sqrt{v_i\gamma + n\lambda^2}|\kappa_i|)^2}. \tag{1.20}$$

Proof. This can be verified by deriving the KKT conditions of the optimization problem in (1.18). The details are moved to Appendix for brevity. □

The results in [14] are weaker because they require a fixed sampling distribution p throughout all iterations. Here we allow adaptive sampling probabilities as in (1.19), which enables the algorithm to utilize the data information more effectively, and hence we have a better convergence rate. Furthermore, the optimal probabilities found in [13] can be only applied to a quadratic loss function, whereas our results are more general because the optimal probabilities in (1.19) can used whenever the loss functions are convex, or when individual loss functions are non-convex but the average of the loss functions is convex (see Section 1.3.2).

Before proceeding with the convergence theory we define several constants. Let

$$C_0 \overset{\text{def}}{=} \tfrac{1}{n}\|\alpha^{(0)} - \alpha^*\|^2 + \gamma\|w^{(0)} - w^*\|^2, \tag{1.21}$$

where γ is defined in (1.12). Note that C_0 in (1.21) is equivalent to the value of the potential function (1.13) at iteration $t = 0$, i.e., $C_0 \equiv D^{(0)}$. Moreover, let

$$M \overset{\text{def}}{=} Q\Big(1 + \frac{\gamma Q}{\lambda^2 n}\Big) \quad \text{where} \quad Q \overset{\text{def}}{=} \frac{1}{n}\sum_{i=1}^n \|x_i\|^2 \overset{(1.14)}{=} \frac{1}{n}\sum_{i=1}^n v_i. \tag{1.22}$$

Now we have the following theorem.

Theorem 1.3.3. *Let $\tilde{L}$, $\kappa_i^{(t)}$, γ, $D^{(t)}$, v_i, C_0 and Q be as defined in (1.5), (1.7), (1.12), (1.13), (1.14), (1.21) and (1.22), respectively. Suppose that ϕ_i is $\tilde{L}$-smooth and convex for all $i \in [n]$, let $\theta^{(t)} \in (0,1)$ be decided by (1.20) for all $t \geq 0$ and let p^* be defined via (1.19). Then, setting $p^{(t)} = p^*$ at every iteration $t \geq 0$ of Algorithm 1.1, gives*

$$\mathbb{E}[D^{(t+1)}|\alpha^{(t)}] \leq (1-\theta^*)D^{(t)}, \tag{1.23}$$

where

$$\theta^* \stackrel{def}{=} \frac{n\lambda^2}{\sum_{i=1}^n (v_i\gamma + n\lambda^2)} \leq \theta^{(t)}. \tag{1.24}$$

Moreover, for $\epsilon > 0$, if

$$T \geq \left(n + \frac{\tilde{L}Q}{\lambda}\right) \log\left(\frac{(\lambda+L)C_0}{2\lambda\tilde{L}\epsilon}\right), \tag{1.25}$$

then $\mathbb{E}[P(w^{(T)}) - P(w^*)] \leq \epsilon$.

Similar to [83], we have the following corollary which bounds the quantity $\mathbb{E}[\|\frac{1}{np_i}\kappa_i^{(t)}x_i\|^2]$ in terms of the sub-optimality of the points $\alpha^{(t)}$ and $w^{(t)}$ by using optimal probabilities.

Corollary 1.3.4. *Let the conditions of Theorem 1.3.3 hold. Then at every iteration $t \geq 0$ of Algorithm 1.1,*

$$\mathbb{E}\left[\left\|\frac{\kappa_i^{(t)}x_i}{np_i}\right\|^2 |\alpha^{(t-1)}\right] \leq 2M(\mathbb{E}[\|\alpha^{(t)} - \alpha^*\|^2|\alpha^{(t-1)}] + L\mathbb{E}[\|w^{(t)} - w^*\|^2|\alpha^{(t-1)}]).$$

Note that Theorem 1.3.3 can be used to show that both $\mathbb{E}[\|\alpha^{(t)} - \alpha^*\|^2]$ and $\mathbb{E}[\|w^{(t)} - w^*\|^2]$ go to zero as $e^{-\theta^* t}$. We can then show that $\mathbb{E}[\|\frac{1}{np_i}\kappa_i^{(t)}x_i\|^2] \leq \epsilon$ as long as $t \geq \tilde{O}(\frac{1}{\theta^*}\log(\frac{1}{\epsilon}))$. Furthermore, we achieve the same variance reduction rate as shown in [83], i.e., $\mathbb{E}[\|\frac{1}{np_i}\kappa_i^{(t)}x_i\|^2] \sim \tilde{O}(\|\kappa^{(t)}\|^2)$.

For the dual free SDCA algorithm in [83] where uniform sampling is adopted, the parameter θ should be set to at most $\min \frac{\lambda}{\lambda n + \tilde{L}}$, where $\tilde{L} \geq \max_i v_i \cdot L$. However, from Corollary 1.3.3, we know that this θ is smaller than θ^*, so dual free SDCA will have a slower convergence rate than our algorithm. In [14], where they use a fixed probability distribution p_i for sampling of coordinates, they must choose θ less than or equal to $\min_i \frac{p_i n\lambda}{L_i v_i + n\lambda}$. This is consistent with [83] where $p_i = 1/n$ for all $i \in [n]$. With respect to our adfSDCA Algorithm

1.1, at any iteration t, we have that $\theta^{(t)}$ is greater than or equal to θ^*, which again implies that our convergence results are better.

1.3.2 Case II: The average of the loss functions is convex

Here we follow the analysis in [83] and consider the case where individual loss functions $\phi_i(\cdot)$ for $i \in [n]$ are allowed to be nonconvex as long as the average $\frac{1}{n}\sum_{j=1}^n \phi_j(\cdot)$ is convex. First we define several parameters that are analogous to the ones used in Section 1.3.1. Let

$$\bar{\gamma} \stackrel{\text{def}}{=} \frac{1}{n}\sum_{i=1}^n L_i^2, \tag{1.26}$$

where L_i is given in (1.4), and define the following potential function. For all iterations $t \geq 0$, let

$$\bar{D}^{(t)} \stackrel{\text{def}}{=} \frac{1}{n}\|\alpha^{(t)} - \alpha^*\|^2 + \bar{\gamma}\|w^{(t)} - w^*\|^2. \tag{1.27}$$

We also define the following constants

$$\bar{C}_0 \stackrel{\text{def}}{=} \frac{1}{n}\|\alpha^{(0)} - \alpha^*\|^2 + \bar{\gamma}\|w^{(0)} - w^*\|^2, \tag{1.28}$$

and

$$\bar{M} \stackrel{\text{def}}{=} Q\left(1 + \frac{\bar{\gamma}Q}{\lambda^2 n}\right). \tag{1.29}$$

Then we have the following theoretical results.

Lemma 1.3.5. *Let L_i, $\kappa_i^{(t)}$, $\bar{\gamma}$, $\bar{D}^{(t)}$, and v_i be as defined in* (1.4), (1.7), (1.26), (1.27) *and* (1.14)*, respectively. Suppose that every $\phi_i, i \in [n]$ is L_i-smooth and that the average of the n loss functions $\frac{1}{n}\sum_{i=1}^n \phi_i(w^T x_i)$ is convex. Let $\theta \in (0,1)$. Then at every iteration $t \geq 0$ of Algorithm 1.1, a probability distribution $p^{(t)}$ that satisfies Definition 1.2.2 is generated and*

$$\mathbb{E}[\bar{D}^{(t+1)}|\alpha^{(t)}] - (1-\theta)\bar{D}^{(t)} \leq \sum_{i=1}^n \left(-\frac{\theta}{n}\left(1 - \frac{\theta}{p_i^{(t)}}\right) + \frac{\theta^2 v_i \bar{\gamma}}{n^2\lambda^2 p_i^{(t)}}\right)(\kappa_i^{(t)})^2. \tag{1.30}$$

Theorem 1.3.6. *Let L, $\kappa_i^{(t)}$, $\bar{\gamma}$ $\bar{D}^{(t)}$, v_i, and $\bar{C}_0$ be as defined in* (1.5), (1.7), (1.26), (1.27), (1.14)*, and* (1.28) *respectively. Suppose that every $\phi_i, i \in [n]$ is L_i-smooth and that the*

average of the n loss functions $\frac{1}{n}\sum_{i=1}^{n}\phi_i(w^T x_i)$ is convex. Let $\theta^{(t)} \in (0,1)$ using (1.20) for all $t \geq 0$ and let p^* be defined via (1.19). Then, setting $p^{(t)} = p^*$ at every iteration $t \geq 0$ of Algorithm 1.1, gives

$$\mathbb{E}\big[\bar{D}^{(t+1)}|\alpha^{(t)}\big] \leq (1-\theta^*)\bar{D}^{(t)}, \tag{1.31}$$

where

$$\theta^* = \frac{n\lambda^2}{\sum_{i=1}^{n}(v_i\bar{\gamma} + n\lambda^2)} \leq \theta^{(t)}.$$

Furthermore, for $\epsilon > 0$, if

$$T \geq \left(n + \frac{\bar{\gamma}Q}{\lambda^2}\right)\log\left(\frac{(\lambda+L)\bar{C}_0}{2\bar{\gamma}\epsilon}\right), \tag{1.32}$$

then $\mathbb{E}[P(w^{(T)}) - P(w^*)] \leq \epsilon$.

We remark that, $L_i \leq L$ for all $i \in [n]$, so $\bar{\gamma} \leq L^2$, which means that a conservative complexity bound is

$$T \geq \left(n + \frac{L^2 Q}{\lambda^2}\right)\log\left(\frac{(\lambda+L)\bar{C}_0}{2\bar{\gamma}\epsilon}\right).$$

We conclude this section with the following corollary.

Corollary 1.3.7. *Let the conditions of Theorem 1.3.6 hold and let $\bar{M}$ be defined in (1.29). Then at every iteration $t \geq 0$ of Algorithm 1.1,*

$$\mathbb{E}\left[\left\|\frac{\kappa_i^{(t)} x_i}{np_i}\right\|^2 |\alpha^{(t-1)}\right] \leq 2\bar{M}(\mathbb{E}[\|\alpha^{(t)} - \alpha^*\|^2|\alpha^{(t-1)}] + L\mathbb{E}[\|w^{(t)} - w^*\|^2|\alpha^{(t-1)}]).$$

1.4 Heuristic adfSDCA

One of the disadvantages of Algorithm 1.1 is that it is necessary to update the entire probability distribution $p \sim \kappa$ at each iteration, i.e., every time a single coordinate is updated the probability distribution is also updated. Note that if the data are sparse and coordinate i is sampled during iteration t, then, one need only update probabilities p_j for which $x_j^T x_i \neq 0$; unfortunately for some datasets this can still be expensive. In order to overcome

this shortfall we follow the recent work in [13] and present a heuristic algorithm that allows the probabilities to be updated less frequently and in a computationally inexpensive way. The process works as follows. At the beginning of each epoch the (full/exact) nonuniform probability distribution is computed, and this remains fixed for the next n coordinate updates, i.e., it is fixed for the rest of that epoch. During that same epoch, if coordinate i is sampled (and thus updated) the probability p_i associated with that coordinate is reduced (it is shrunk by $p_i \leftarrow p_i/s$), where s is the shrinkage parameter. The intuition behind this procedure is that, if coordinate i is updated then the dual residue $|\kappa_i|$ associated with that coordinate will decrease. Thus, there will be little benefit (in terms of reducing the suboptimality of the current iterate) in sampling and updating that same coordinate i again. To avoid choosing coordinate i in the next iteration, we shrink the probability p_i associated with it, i.e., we reduce the probability by a factor of $1/s$. Moreover, shrinking the coordinate is less computationally expensive than recomputing the full adaptive probability distribution from scratch, and so we anticipate a decrease in the overall running time if we use this heuristic strategy, compared with the standard adfSDCA algorithm. This procedure is stated formally in Algorithm 1.2. Note that Algorithm 1.2 does not fit the theory established in Section 1.3. Nonetheless, we have observed convergence in practice and a good numerical performance when using this strategy (see the numerical experiments in Section 1.6).

Algorithm 1.2 Heuristic Adaptive Dual Free SDCA (adfSDCA+)

Input: Data: $\{x_i, \phi_i\}_{i=1}^n$, probability shrink parameter s
Initialization: Choose $\alpha^{(0)} \in \mathbb{R}^n$
Set $w^{(0)} = \frac{1}{\lambda n}\sum_{i=1}^n \alpha_i^{(0)} x_i$
for $t = 0, 1, 2, \ldots$ **do**
  if $\mod(t, n) == 0$ **then**
    Calculate dual residue $\kappa_i^{(t)} = \phi_i'(x_i^T w^{(t)}) + \alpha_i^{(t)}$, for all $i \in [n]$
    Generating adapted probabilities distribution $p^{(t)} \sim \kappa^{(t)}$
  end if
  Select coordinate i from $[n]$ according to $p^{(t)}$
  Set step-size $\theta^{(t)} \in (0, 1)$ as in (1.20)
  Update: $\alpha_i^{(t+1)} = \alpha_i^{(t)} - \theta^{(t)}(p_i^{(t)})^{-1}\kappa_i^{(t)}$
  Update: $w^{(t+1)} = w^{(t)} - \theta^{(t)}(n\lambda p_i^{(t)})^{-1}\kappa_i^{(t)} x_i$
  Update: $p_i^{(t+1)} = p_i^{(t)}/s$
end for

1.5 Mini-batch adfSDCA

In this section we propose a mini-batch variant of Algorithm 1.1. Before doing so, we stress that sampling a mini-batch non-uniformly is not easy. We first focus on the task of generating non-uniform random samples and then we will present our minibatch algorithm.

1.5.1 Efficient single coordinate sampling

Before considering mini-batch sampling, we first show how to sample a single coordinate from a non-uniform distribution. Note that only discrete distributions are considered here.

There are multiple approaches that can be taken in this case. One naïve approach is to consider the Cumulative Distribution Function (CDF) of p, because a CDF can be computing in $O(n)$ time complexity and it also takes $O(n)$ time complexity to make a decision. One can also use a better data structure (e.g. a binary search tree) to reduce the decision cost to $O(\log n)$ time complexity, although the cost to set up the tree is $O(n \log n)$. Some more advanced approaches like the so-called alias method of [44] can be used to sample a single coordinate in only $O(1)$, i.e., sampling a single coordinate can be done in constant time but with a cost of $O(n)$ setup time. The alias method works based on the fact that any n-valued distribution can be written as a mixture of n Bernoulli distributions.

In this chapter we choose two sampling update strategies, one each for Algorithms 1.1 and 1.2. For adfSDCA in Algorithm 1.1 the probability distribution must be recalculated at every iteration, so we use the alias method, which is highly efficient. The heuristic approach in Algorithm 1.2 is a strategy that only alters the probability of a single coordinate (e.g. $p_i = p_i/s$) in each iteration. In this second case it is relatively expensive to use the alias method due to the linear time cost to update the alias structure, so instead we build a binary tree when the algorithm is initialized so that the update complexity reduces to $O(\log(n))$.

1.5.2 Nonuniform Mini-batch Sampling

Many randomized coordinate descent type algorithms utilize a sampling scheme that assigns every subset of $[n]$ a probability p_S, where $S \in 2^{[n]}$. In this section, we consider a particular type of sampling called a *mini-batch* sampling that is defined as follows.

Definition 1.5.1. *A sampling $\hat{S}$ is called a mini-batch sampling, with batchsize b, consistent with the given marginal distribution $q := (q_1, \ldots, q_n)^T$, if the following conditions hold:*

1. $|S| = b$;
2. $q_i \overset{def}{=} \sum_{S \in \hat{S}} P(\{S : i \in S\}) = bp_i$,

where $P(\{S : i \in S\})$ represents the probability of mini-batch sampling S containing the coordinate i.

Here we are going to derive a proper sampling strategy over coordinate i such that $i \in S \in \hat{S}$ and Definition 1.5.1 is satisfied. Note that we study samplings $\hat{S}$ that are *non-uniform* since we allow q_i to vary with i. The motivation to design such samplings arises from the fact that we wish to make use of the optimal probabilities that were studied in Section 1.3.

We make several remarks about non-uniform mini-batch samplings below.

1. For a given probability distribution p, one can derive a corresponding mini-batch sampling only if we have $p_i \leq \frac{1}{b}$ for all $i \in [n]$. This is obvious in the sense that $q_i = bp_i = \sum_{S \in \hat{S}} P(\{S : i \in S\}) \leq \sum_{S \in \hat{S}} P(S) = 1$.

2. For a given probability distribution p and a batch size b, the mini-batch sampling may not be unique and it may not be proper, see for example [74]. (A proper sampling is a sampling for which any subset of size b must have a *positive* probability of being sampled.)

In Algorithm 1.3 we describe an approach that we used to generate a non-uniform mini-batch sampling of batchsize b from a given marginal distribution q. Without loss of generality, we assume that the $q_i \in (0, 1)$ for $i \in [n]$ are sorted from largest to smallest.

We now state several facts about Algorithm 1.3.

1. Algorithm 1.3 will terminate in at most n iterations. This is because the update rules for q_i (which depend on r_k at each iteration), ensure that at least one q_i will reduce to become equal to some $q_j < q_i$ (i.e., either $q_{i^{k+1}-1} = q_b$ or $q_{j^{k+1}+1} = q_b$) and since there are n coordinates in total, after at most n iteration it must hold that $q_i = q_j$ for all

Algorithm 1.3 Non-uniform mini-batch sampling

Input: Marginal distribution $q \in \mathbb{R}^n$ with $q_i \in (0,1)\ \forall i \in [n]$ and batchsize b such that $\sum_{i=1}^n q_i = b$. Define $q_{n+1} = 0$

Output: A mini-batch sampling S (Definition 1.5.1)

Initialization: Index set $i, j \in \mathbb{N}^n$, and set $k = 1$.

for $k = 1, \ldots, n$ **do**

$i^k = \min_i\{i : p_i = q_b\}, j^k = \max_i\{i : p_i = q_b\}$

Obtain r_k:

$$r_k = \begin{cases} \min\left\{\frac{j^k - i^k + 1}{j^k - b}(q_{i^k - 1} - q_b), \frac{j^k - i^k + 1}{b - i^k + 1}(q_b - q_{j^k + 1})\right\}, & i^k > 1 \\ \frac{j}{b}(q_b - q_{j^k + 1}), & i^k = 1 \end{cases} \tag{1.33}$$

Update q_i:

$$q_i = \begin{cases} q_i - r_k, & i \in [0, i^k - 1], \\ q_i - \frac{b - i^k + 1}{j^k - i^k + 1} r_k, & i \in [i^k, j^k] \end{cases} \tag{1.34}$$

Terminate if $q = 0$, and set $m = k$

end for

Select $K \in [m]$ randomly with discrete distribution $(r_1, \ldots, r_m)$

Choose $b - i^K + 1$ coordinates uniformly at random from i^K to j^K, denote it by W

$S = \{1, \ldots, i^K - 1\} \cup W$

$i, j \in [n]$. Note that if the algorithm begins with $q_i = q_j$ for all $i, j \in [n]$, which implies a uniform marginal distribution, the algorithm will terminated in a single step.

2. For Algorithm 1.3 we must have $\sum_{i=1}^m r_i = 1$, where we assume that the algorithm terminates at iteration $m \in [1, n]$, since overall we have $\sum_{i=1}^m b r_i = \sum_{i=1}^n q_i = b$.

3. Algorithm 1.3 will always generate a proper sampling because when it terminates, the situation $p_i = p_j > 0$, for all $i \neq j$, will always hold. Thus, any subset of size b has a positive probability of being sampled.

4. It can be shown that this algorithm works on an arbitrary given marginal probabilities as long as $q_i \in (0,1)$, for all $i \in [n]$.

Figure 1.1 is a sample illustration of Algorithm 1.3, where we have a marginal distribution for 4 coordinates given by $(0.8, 0.6, 0.4, 0.2)^T$ and we set the batchsize to be $b = 2$. Then, the algorithm is run and finds r to be $(0.2, 0.4, 0.4)^T$. Afterwards, with probability $r_1 = 0.2$, we will sample 2-coordinates from $(1, 2)$. With probability $r_2 = 0.4$, we will sample 2-coordinates which has (1) for sure and the other coordinate is chosen from $(2, 3)$ uniformly

at random and with probability $r_3 = 0.4$, we will sample 2-coordinates from $(1, 2, 3, 4)$ uniformly at random.

Note that, here we only need to perform two kinds of operations. The first one is to sample a single coordinate from distribution d (see Section 1.5.1), and the second is to sample batches from a uniform distribution (see for example [74]).

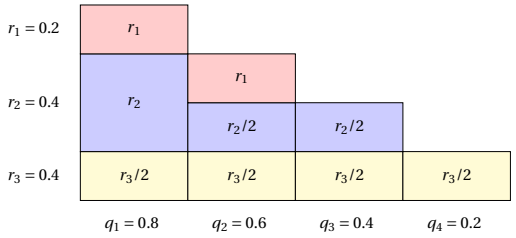


Figure 1.1: Toy demo illustrating how to obtain a non-uniform mini-batch sampling with batch size $b = 2$ from $n = 4$ coordinates.

1.5.3 Mini-batch adfSDCA algorithm

Here we describe a new adfSDCA algorithm that uses a mini-batch scheme. The algorithm is called mini-batch adfSDCA and is presented below as Algorithm 1.4.

Algorithm 1.4 Mini-Batch adfSDCA

Input: Data: $\{x_i, \phi_i\}_{i=1}^n$
Initialization: Choose $\alpha^{(0)} \in \mathbb{R}^n$ and set batchsize b
for $t = 0, 1, 2, \ldots$ **do**
  Calculate dual residue $\kappa_i^{(t)} = \phi_i'(x_i^T w^{(t)}) + \alpha_i^{(t)}$, for all $i \in [n]$
  Generate the adaptive probability distribution $p^{(t)} \sim \kappa^{(t)}$
  Choose mini-batch $S \subset [n]$ of size b according to probabilities distribution $p^{(t)}$
  Set step-size $\theta^{(t)} \in (0, 1)$ as in (A.36)
  for $i \in S$ **do**
    Update: $\alpha_i^{(t+1)} = \alpha_i^{(t)} - \theta^{(t)}(bp_i^{(t)})^{-1}\kappa_i^{(t)}$
  end for
  Update: $w^{(t+1)} = w^{(t)} - \sum_{i \in S} \theta^{(t)}(n\lambda bp_i^{(t)})^{-1}\kappa_i^{(t)} x_i$
end for

Briefly, Algorithm 1.4 works as follows. At iteration t, adaptive probabilities are generated in the same way as for Algorithm 1.1. Then, instead of updating only one coordinate, a mini-batch S of size $b \geq 1$ is chosen that is consistent with the adaptive probabilities. Next, the dual variables $\alpha_i^{(t)}, i \in S$ are updated, and finally the primal variable w is updated according to the primal-dual relation (1.6).

In the next section we will provide a convergence guarantee for Algorithm 1.4. As was discussed in Section 1.3, theoretical results are detailed under two different assumptions on the type of loss function: (i) all loss function are convex; and (ii) individual loss functions may be non-convex but the average over all loss functions is convex.

1.5.4 Expected Separable Overapproximation

Here we make use of the Expected Separable Overapproximation (ESO) theory introduced in [74] and further extended, for example, in [72]. The ESO definition is stated below.

Definition 1.5.2 (Expected Separable Overapproximation, [72])**.** *Let $\hat{S}$ be a sampling with marginal distribution $q = (q_1, \cdots, q_n)^T$. Then we say that the function f admits a v-ESO with respect to the sampling $\hat{S}$ if $\forall x, h \in \mathbb{R}^n$, we have $v_1, \ldots, v_n > 0$, such that the following inequality holds $\mathbb{E}[f(x + h_{[\hat{S}]})] \leq f(x) + \sum_{i=1}^n q_i(\nabla_i f(x)h_i + \frac{1}{2}v_i h_i^2)$.*

Remark 1.5.3. *Note that, here we do not assume that $\hat{S}$ is a uniform sampling, i.e., we do not assume that $q_i = q_j$ for all $i, j \in [n]$.*

The ESO inequality is useful in this chapter because the parameter v plays an important role when setting a suitable stepsize θ in our algorithm. Consequently, this also influences our complexity result, which depends on the sampling $\hat{S}$. For the proof of Theorem 1.5.5 (which will be stated in next subsection), the following is useful. Let $f(x) = \frac{1}{2}\|Ax\|^2$, where $A = (x_1, \ldots, x_n)$. We say that $f(x)$ admits a v-ESO if the following inequality holds

$$\mathbb{E}[\|Ah_{\hat{S}}\|^2] \leq \sum_{i=1}^n v_i q_i h_i^2. \tag{1.35}$$

To derive the parameter v we will make use of the following theorem.

Theorem 1.5.4 ([72])**.** *Let f satisfy the following assumption $f(x+h) \leq f(x)+\langle \nabla f(x), h\rangle + \frac{1}{2}h^T A^T A h^T$, where A is some matrix. Then, for a given sampling $\hat{S}$, f admits a v-ESO, where v is defined by $v_i = \min\{\lambda'(\mathbf{P}(\hat{S})), \lambda'(A^T A)\}\sum_{j=1}^m A_{ji}^2, i \in [n]$.*

Here $\mathbf{P}(\hat{S})$ is called a sampling matrix (see [74]) where element p_{ij} is defined to be $p_{ij} = \sum_{\{i,j\}\in S, S\in\hat{S}} P(S)$. For any matrix M, $\lambda'(M)$ denotes the maximal regularized eigenvalue of

M, i.e., $\lambda'(M) = \max_{\|h\|=1}\{h^T M h : \sum_{i=1}^n M_{ii} h_i^2 \leq 1\}$. We may now apply Theorem 1.5.4 because $f(x) = \frac{1}{2}\|Ax\|^2$ satisfies its assumption. Note that in our mini-batch setting, we have $P_{S\in\hat{S}}(|S| = b) = 1$, so we obtain $\lambda'(\mathbf{P}(\hat{S})) \leq b$ (Theorem 4.1 in [72]). In terms of $\lambda'(A^T A)$, note that $\lambda'(A^T A) = \lambda'(\sum_{j=1}^m x_j x_j^T) \leq \max_j \lambda'(x_j x_j^T) = \max_j |J_j|$, where $|J_j|$ is number of non-zero elements of x_j for each j. Then, a conservative choice from Theorem 1.5.4 that satisfies (1.35) is

$$v_i' = \min\{b, \max_j |J_j|\}\|x_i\|^2, \qquad i \in [n]. \tag{1.36}$$

Now we are ready to give our complexity result for mini-batch adfSDCA (Algorithm 1.4). Note that we use the same notation as that established in Section 1.3 and we also define

$$Q' \stackrel{\text{def}}{=} \frac{1}{n}\sum_{i=1}^n v_i'. \tag{1.37}$$

Theorem 1.5.5. *Let $\tilde{L}$, $\kappa_i^{(t)}$, γ $D^{(t)}$, v_i', C_0 and Q' be as defined in* (1.5), (1.7), (1.12), (1.13), (1.36), (1.21) *and* (1.37)*, respectively. Suppose that ϕ_i is L-smooth and convex for all $i \in [n]$. Then, at every iteration $t \geq 0$ of Algorithm 1.4, run with batchsize b we have*

$$\mathbb{E}[D^{(t+1)}|\alpha^{(t)}] \leq (1 - \theta^*)D^{(t)}, \tag{1.38}$$

where $\theta^* = \frac{n\lambda^2 b}{\sum_{i=1}^n (v_i'\gamma + n\lambda^2)}$. Moreover, it follows that whenever

$$T \geq \left(\frac{n}{b} + \frac{\tilde{L}Q'}{b\lambda}\right)\log\left(\frac{(\lambda + \tilde{L})C_0}{\lambda\tilde{L}\epsilon}\right), \tag{1.39}$$

we have that $\mathbb{E}[P(w^{(T)} - P(w^*))] \leq \epsilon$.

It is also possible to derive a complexity result in the case when the *average* of the n loss functions is convex. The theorem is stated now.

Theorem 1.5.6. *Let L, $\kappa_i^{(t)}$, $\bar{\gamma}$ $\bar{D}^{(t)}$, v_i', $\bar{C}_0$ and Q' be as defined in* (1.5), (1.7), (1.26), (1.27), (1.36), (1.28) *and* (1.37) *respectively. Suppose that every $\phi_i, i \in [n]$ is L_i-smooth and that the average of the n loss functions $\frac{1}{n}\sum_{i=1}^n \phi_i(w^T x_i)$ is convex. Then, at every iteration*

$t \geq 0$ of Algorithm 1.4, run with batchsize b, we have

$$\mathbb{E}[\bar{D}^{(t+1)}|\alpha^{(t)}] \leq (1-\theta^*)\bar{D}^{(t)}, \tag{1.40}$$

where $\theta^* = \frac{n\lambda^2 b}{\sum_{i=1}^n (v_i'\bar{\gamma}+n\lambda^2)}$. Moreover, it follows that whenever

$$T \geq \left(\frac{n}{b} + \frac{Q'\frac{1}{n}\sum_{i=1}^n L_i^2}{b\lambda}\right)\log\left(\frac{(\lambda+\tilde{L})\bar{C}_0}{\bar{\gamma}\epsilon}\right), \tag{1.41}$$

we have that $\mathbb{E}[P(w^{(T)}) - P(w^*)] \leq \epsilon$.

These theorems show that in worst case (by setting $b = 1$), this mini-batch scheme shares the same complexity performance as the serial adfSDCA approach (recall Section 1.2). However, when the batch-size b is larger, Algorithm 1.4 converges in fewer iterations. This behavior will be confirmed computationally in the numerical results given in Section 1.6.

1.6 Numerical experiments

Here we present numerical experiments to demonstrate the practical performance of the adfSDCA algorithm. Throughout these experiments we used two loss functions, quadratic loss $\phi_i(w^T x_i) = \frac{1}{2}(w^T x_i - y_i)^2$ and logistic loss $\phi_i(w^T x_i) = \log(1+\exp(-y_i w^T x_i))$. Note that these two losses have Lipschitz gradient. The regularization parameter λ in (1.1) is set to be $1/\sqrt{n}$, where n is the number of samples of the dataset. The experiments were run using datasets from the standard library of test problems (see [11] and `http://www.csie.ntu.edu.tw/~cjlin/libsvm`), as summarized in Table 1.1.

Dataset	#**samples**	#**features**	#**classes**	**sparsity**
mushrooms	8,124	112	2	18.8%
ijcnn1	49,990	22	2	59.1%
rcv1	20,242	47,237	2	0.16%
news20	19,996	1,355,191	2	0.034%

Table 1.1: A list of datasets used in the numerical experiments, see [11].

1.6.1 Comparison for a variety of adfSDCA approaches

In this section we compare the adfSDCA algorithm (Algorithm 1.1) with both dfSCDA, which is a uniform variant of adfSDCA described in [83], and also with Prox-SDCA from [87]. We also report results using Algorithm 1.2, which is a heuristic version of adfSDCA, used with several different shrinkage parameters.

Figure 1.2 compares the evolution of the duality gap for the standard and heuristic variant of our adfSDCA algorithm with the two state-of-the-art algorithms dfSDCA and Prox-SDCA. For these problems both our algorithm variants out-perform the dfSDCA and Prox-SDCA algorithms. Note that this is consistent with our convergence analysis (recall Section 1.3). Now consider the adfSDCA+ algorithm, which was tested using the parameter values $s = 1, 10, 20$. It is clear that adfSDCA+ with $s = 1$ shows the worst performance, which is reasonable because in this case the algorithm only updates the sampling probabilities after each epoch; it is still better than dfSDCA since it utilizes the sub-optimality at the beginning of each epoch. On the other hand, there does not appear to be an obvious difference between adfSDCA+ used with $s = 10$ or $s = 20$ with both variants performing similarly. We see that adfSDCA performs the best overall in terms of the number of passes through the data. However, in practice, even though adfSDCA+ may need more passes through the data to obtain the same sub-optimality as adfSDCA, it requires less computational effort than adfSDCA.

In Figure 1.3, we compare SGD, SVRG, dfSDCA and our proposed adfSDCA(+) algorithm in terms of the number of passes through the data and total running time. For the SGD and SVRG algorithms, the duality gap is not directly computable. Hence, in this numerical experiment, the relative primal objective value $P(w) - P(\hat{w})$ is used as the stopping condition, where $\hat{w}$ is the optimal weight given by the best run among all algorithms. The SGD algorithm is implemented using the same set-up as in [85], where a diminishing step-size is used, and SVRG is implemented following [36].

We remark that for the SVRG algorithm, the user must tune its two hyper-parameters, namely, the number of iterations in the inner loop, and the step-size. Proper tuning of these hyper-parameters is essential to get the best performance from the SVRG algorithm. In this

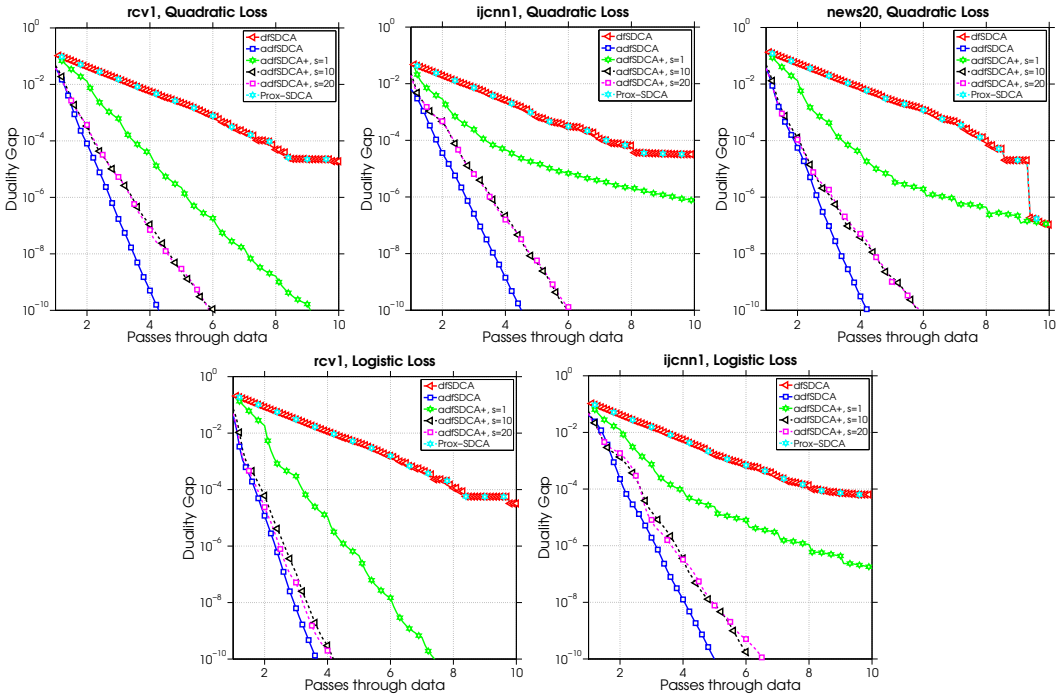


Figure 1.2: A comparison of the number of epochs versus the duality gap for the various algorithms.

experiment, we tuned the hyper-parameters for SVRG, and we used SVRG+ to denote the best performing SVRG variant, and we use m to denote the corresponding 'best' number of inner loop iterations. As a means of comparison, we also plot the performance of the SVRG algorithm using $m/2$ and $2m$ inner loop iterations (i.e., SVRG without optimal tuning).

Figure 1.3 shows that, for the `rcv1` dataset with a quadratic loss, adfSDCA is the best performing algorithm in terms of the number of passes through the data; it is even better than the 'best' tuned SVRG algorithm. For the `ijcnn1` dataset with a quadratic loss, SVRG+, the optimally tuned SVRG algorithm, performs better than the adfSDCA algorithm. However, tuning the hyper-parameters for SVRG is not free, and this is a computational cost that is not required for adfSDCA. This highlights one of the benefits of adfSDCA, which does not require parameter tuning, and the specific step-size needed is given explicitly in Theorem 1.3.1.

We also present plots showing the total running time for these algorithms. We follow the set up in [13], and present the running time results using the heuristic algorithm adfSDCA+ with the shrinkage parameter set to $s = 5$ (see Section 1.4). Recall that the `rcv1` dataset

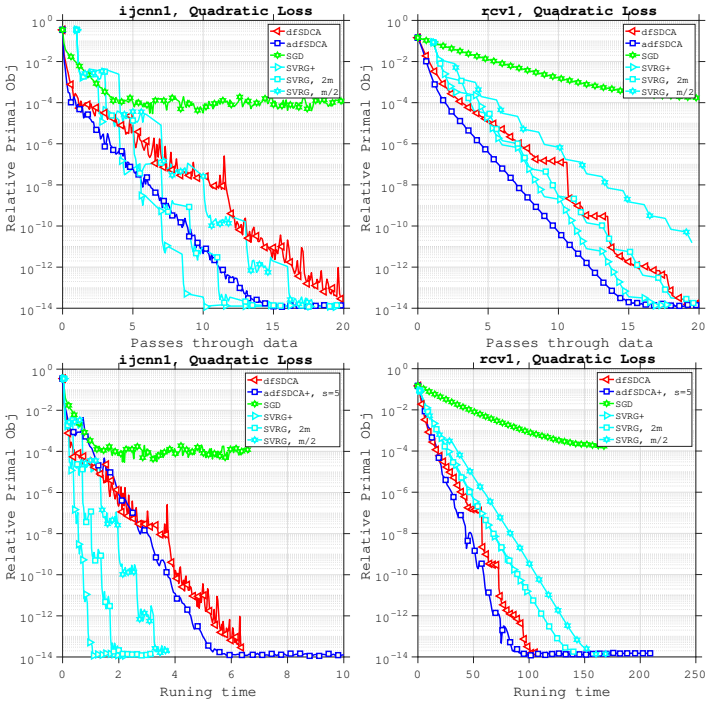


Figure 1.3: A comparison of the number of epochs versus the relative primal object value for SGD, dfSDCA, adfSDCA(+) and SVRG. SVRG+ denotes the parameter-tuned, best performing SVRG algorithm, where m denotes the corresponding number of inner loop iterations. We also show results for the SVRG agorithm using both $m/2$ and $2m$ inner loop iterations, to demonstrate the performance of SVRG without optimal tuning.

has $n = 20,242$ and $d = 47,237$, so the number of samples is comparable to the number of features. For this experiment, Figure 1.3 shows that the total running time needed for adfSDCA+ is much less than SVRG. However, for the `ijcnn1` dataset, SVRG outperforms adfSDCA+ in terms of running time. To gain some insight into why this is happening, recall that the `ijcnn1` dataset has $n = 49,990$ and $d = 22$, so the number of samples is *much more* than the number of features. Note that adfSDCA+ must compute the residuals for each coordinate at every iteration, and because the number of samples is far greater than the number of feature, there is a high running time overhead for this non-uniform sampling of coordinates for adfSDCA+. This suggests that it is beneficial to use adfSDCA when the number of features is comparable with the number of samples.

Figure 1.4 shows the estimated density function of the dual residue $|\kappa^{(t)}|$ after $1, 2, 3, 4$ and 5 epochs for both uniform dfSDCA and our adaptive adfSDCA. One observes that the adaptive scheme is pushing the large residuals towards zero much faster than uniform

dfSDCA. For example, notice that after 2 epochs, almost all residuals are below 0.03 for adfSDCA, whereas for uniform dfSDCA there are still many residuals larger than 0.06. This is evidence that, by using adaptive probabilities we are able to update the coordinate with a high dual residue more often and therefore reduce the sub-optimality much more efficiently.

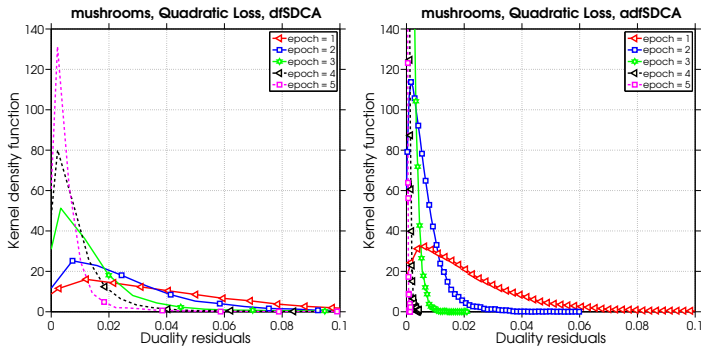


Figure 1.4: Comparing absolute value of dual residuals at each epoch between dfSDCA and adfSDCA.

1.6.2 Mini-batch adfSDCA

Here we investigate the behavior of the mini-batch adfSDCA algorithm (Algorithm 1.4). In particular, we compare the practical performance of mini-batch adfSDCA using different mini-batch sizes b varying from 1 to 32. Note that if $b = 1$, then Algorithm 1.4 is equivalent to the adfSDCA algorithm (Algorithm 1.1). Figures 1.5 shows that, with respect to the different batch sizes, the mini-batch algorithm with each batch size needs roughly the same number of passes through the data to achieve the same sub-optimality. However, when considering the computational time, the larger the batch size is, the faster the convergence will be. Recall that the results in Section 1.5 show that the number of iterations needed by Algorithm 1.4 used with a batch size of b is roughly $1/b$ times the number of iterations needed by adfSDCA. Here we compute the adaptive probabilities every b samples, which leads to roughly the same number of passes through the data to achieve the same sub-optimality.

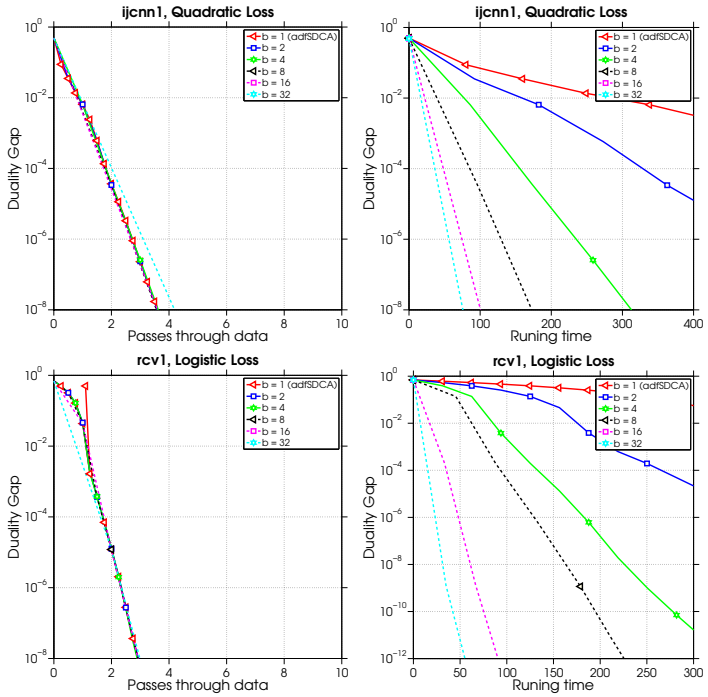


Figure 1.5: Comparing the number of iterations of various batch size on different losses.

1.6.3 adfSDCA for non-convex loss

Here we investigate the behavior of adfSDCA when applied to problems that involve some nonconvex loss functions. We describe the experimental set-up now. Suppose that we have convex loss functions $\phi_i(x_i^T w)$, where $i \in [n]$. Then, it is possible to construct nonconvex loss functions by subtracting a quadratic from each of the convex losses as follows:

$$\bar{\phi}_i(x_i^T w) = \phi_i(x_i^T w) - C_i \|w\|^2. \tag{1.42}$$

Note that if $C_i > 0$ is large enough (up to the Lipschitz gradient constant of $\phi_i(x_i^T w)$), the new loss $\bar{\phi}_i(x_i^T w)$ derived by (1.42) will be nonconvex. On the other hand, if $C_i < 0$, we will have the new loss being strongly convex.

Now, functions of the form (1.42) will satisfy the requirements of Case II in Section 3.2 (i.e., that the individual loss functions can be nonconvex, but that the average over all the losses is convex) as long as some of the hyperparameters C_i are large enough to make (1.42) nonconvex and $\sum_{i=1}^n C_i = 0$. Using this set-up, we present a numerical experiment to show the practical performance of adfSDCA. The quadratic loss is applied in this experiment

due to that the new loss (1.42) would be nonconvex when $C_i > 0$, since the Hessian of each quadratic loss of x_i has the smallest eigenvalue 0. In particular, we let $C_i = 0.01 \times (-1)^i$, where $i \in [n]$. We use the *mushrooms* and *ijcnn1* datasets for this experiment, and because these datasets both have an even number of samples, the property that $\sum_{i=1}^{n} C_i = 0$ will hold. The results of this experiment are shown in Figure 8, where we compare the performance of adfSDCA with respect to the running time and number of passes over the data. Figure 8 shows that adfSDCA performs well on such problems and is able to find an accurate solution (where the duality gap is less than 10^{-10}) in less than 20 passes over the data.

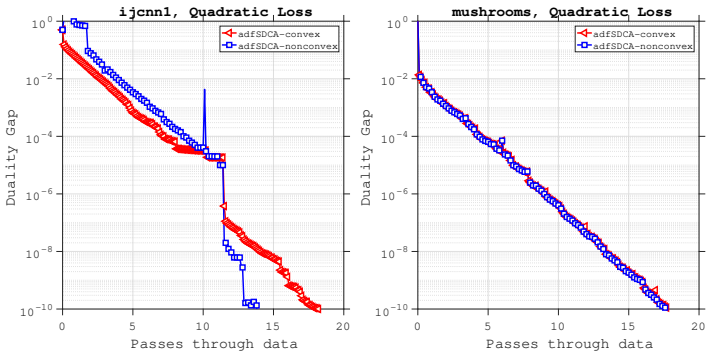


Figure 1.6: Comparing adfSDCA for two cases on quadratic loss.

1.7 Conclusion

In this chapter, we present dual free SDCA variants with adaptive probabilities for Empirical Risk Minimization problems. The theoretical complexity of the proposed methods is analyzed in two cases: when the individual loss functions are all convex and when the average over the losses is convex but individual loss functions may be nonconvex. A heuristic variant of adfSDCA is proposed to reduce the computational effort required and its practical convergence performance is demonstrated via a numerical experiment. We also extend our convergence theory to cover a mini-batch adfSDCA variant and a novel nonuniform sampling strategy for mini-batches is developed. Our experimental results show speedups in terms of the number of passes through the data and/or running time of the proposed methods, when compared with the original dual free SDCA, as well as other state-of-art primal methods.

The numerical experiments related to the use of mini-batches match our theoretical analysis and suggest that using mini-batches is beneficial in practice.

Acknowledgement

We would like to thank Professor Alexander L. Stolyar for his insightful help with Algorithm 1.3.

Chapter 2

Large-scale Distributed Hessian-Free Optimization for Deep Neural Networks

2.1 Introduction

Deep learning has shown great success in many practical applications, such as image classification [43, 89, 27], speech recognition [31, 81, 1], etc. Stochastic gradient descent (SGD), as one of the most well-developed method for training neural network, has been widely used. Besides, there has been plenty of interests in second order methods for training deep networks [51]. The reasons behind these interests are multi-fold. At first, it is generally more substantial to apply weight updates derived from second order methods in terms of optimization aspect, meanwhile, it takes roughly the same time to obtain curvature-vector products [39] as it takes to compute gradient which make it possible to use second order method on large scale model. Furthermore, computing gradient and curvature information on large batch (even whole dataset) can be easily distributed across several nodes. Recent work has also been used to reveal the significance of identifying and escaping saddle point by second order method, which helps prevent the dramatic deceleration of training speed around the saddle point [17].

Line search Newton-CG method (also known as the truncated Newton Method), as one of the practical techniques to achieve second order method on high dimensional optimization, has been studied for decades [65]. Recent work to apply Newton-CG method has been proved as a practical and successful achievement on training deep neural network[51, 39]. Indeed, for Newton-CG method, at each iteration, an approximated Hessian matrix is constructed, and naïve conjugate gradient (CG) method is applied to obtain a descent direction. The naïve CG method is, however, designed to solve positive definite systems, i.e., it requires the approximate Hessian matrix to be positive definite. Otherwise, the CG iteration is terminated as soon as a negative curvature direction is generated. Note that Newton-CG method does not require explicit knowledge of Hessian matrix, and it requires only the Hessian-vector product for any given vector. One special case for using Hessian-vector product is to train deep neural network, also known as Hessian-free optimization, and such Hessian-free optimization is exactly used in Marten's HF [51] methods.

As discussed in [17], identifying and escaping saddle points significantly improve training performance. This implies the necessity to use negative curvature direction. Conventionally with Newton-CG methods, the negative curvature direction is simply ignored, which may lead to unsatisfactory training. In this chapter, we highlight the importance of the using of negative curvature direction and the its impact therein on training, with a small demo example. we go to further derive ways to find negative curvature direction and propose a novel algorithm to use such negative curvature effectively.

Moreover, it is well known that traditional SGD method is inherently sequential and becomes very expensive (time-to-train) to apply on very large data sets. More detail discussion can be found in [102], wherein Momentum SGD (MSGD) [92], ASGD and MVASGD [70], are considered as alternatives. However, it is shown that these methods have limited scaling potential, due to the limited concurrency. However, unlike SGD, Hessian-free methods (in this work, we are focus on full gradient and stochastic Hessian-vector product evaluation) can be distributed naturally, allow for large mini-batch sizes (increased parallelism) while improving convergence rate and also the better the quality of solution - we are therefore motivated to develop a distributed variant of Hessian-free optimization.

In this chapter, we explore the Hessian-free methods to develop more robust and scalable

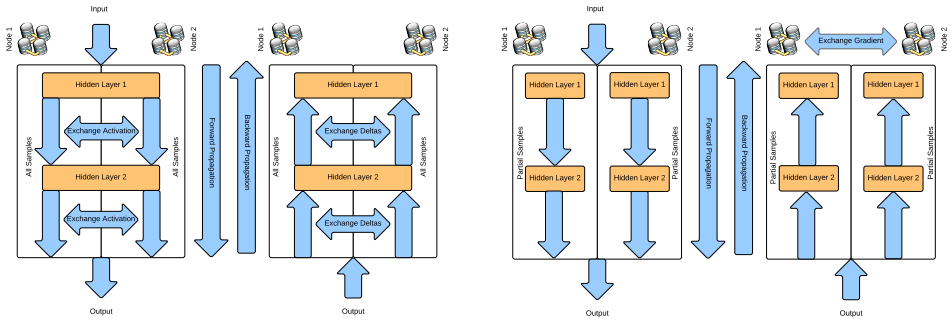


Figure 2.1: Model (left) and data (right) parallelism.

solver for deep learning. We discuss novel ways to utilize negative curvature information to accelerate training speed. This is different with original Marten's HF, where the negative curvature is ignored by either using Gauss-Newton Hessian approximation or truncated Newton method. We perform experimental evaluations on two datasets without distortions or pre-training: hand written digits recognition (MNIST) and speech recognition (TIMIT).

Additionally, we explore Hessian-free methods in a distributed context. Its potential scaling property is discussed, showcasing scaling potential of distributed Hessian-free method and how it allows taking advantage of more computing resources without being limited by the expensive communication.

2.2 Deep Neural Network in Distributed Environment

Training DNNs can be parallelized using the following two strategies - model parallelism (we split weights across many computing nodes) and data parallelism (when the data is partitioned across nodes).

Let us briefly explain how data and model parallelism works and what are the bottlenecks if SGD is implemented in a distributed way choosing either parallelism approach as depicted in Figure 2.1.

Model Parallelism. In the model parallelism the weights of network are split across N nodes. In one SGD iteration all nodes work on the same data but each is responsible only for some of the features. Hence after each layer they have to synchronize to have the activations needed for the portion of the model they have for in next layer. For the backward

pass they have to also synchronize after each layer and exchange the δ's used to compute gradients. After gradients are computed they can be applied to weights stored locally.

If a mini-batch of size b is used and the weights for hidden layer have dimensions $d_1 \times d_2$, then each node (if split equally) will have to store $\frac{d_1 \times d_2}{N}$ floats. The total amount of data exchanged over network for this single layer is $d_1 \times b$. If we consider a deeper network with dimensions $d_1, d_2, \ldots, d_l$ then the total number of floats to be exchanged in one epoch of SGD is approximately $2 \times \frac{n}{b} \times b \sum_i d_i$ and total number of communications (synchronizations) needed per one epoch is $2 \times l \times \frac{n}{b}$.

Data Parallelism. The other natural way how to implement distributed SGD for DNN is to make a copy of weights on each node and split the data across N nodes, where each node owns roughly n/N samples. When a batch of size b is chosen, on each node only $\frac{b}{N}$ samples are propagated using forward and backward pass. Then the gradients are reduced and applied to update weights. We then have to make sure that after each iteration of SGD all weights are again synchronized. In terms of data sent over the network, in each iteration of SGD we have to reduce the gradients and broadcast them back. Hence amount of data to be send over the network in one epoch is $\frac{n}{b} \times \log(N) \times \sum_{i=1}^{l} d_0 \times d_i$, where $d_0 = d$ is the dimension of the input samples. Total number of MPI calls per epoch is hence only $\frac{n}{b} \times 2$ which is considerably smaller then for the model parallelism approach.

Limits of SGD. As it can be seen from the estimates for amount of communication and the frequency of communication, choosing large value of b in the data parallelism will minimize communication and for data parallelism also amount of data sent. However, as it was observed e.g. in [93] SGD (even for convex problem) can benefit from mini-batch only for small batch size b. After increasing b above a critical value $\tilde{b}$, number of iterations needed to achieve a desired accuracy will not be decreased much if the batch size $b > \tilde{b}$. Quite naturally this can be observed also for training DNN [16, 102, 88].

Benefits of Distributed HF. As we will show in following sections, distributed HF need less synchronizations/communications per epoch. SGD requires synchronization after each update (mini-batch). In distributed HF, we only need synchronize once for one full-gradient computing and other several times (much less than what we need of SGD, considering its limitation of using mini-batch size) which is related to number of CG iterations.

2.3 Distributed Hessian-free Optimization Algorithms

In this Section we describe a distributed Hessian-free algorithms. We assume that the size of the model is not huge and hence we choose data parallelism paradigm. We assume that the samples are split equally across K computing nodes (MPI processes).

2.3.1 Distributed HF optimization framework

Within this Hessian-free optimization approach, for the sake of completeness, we first state the general Hessian-free optimization method [51] in Algorithm 2.1. Here $\theta \in \mathbb{R}^N$ is the

Algorithm 2.1 The Hessian-free optimization method

1: **for** $k = 1, 2, \ldots$ **do**
2: $g_k = \nabla f(\theta_k)$
3: Compute/adjust damping parameter λ
4: Define $B_k(d) = H(\theta_k)d + \lambda d$
5: $p_k = \text{CG-Minimize}(B_k, -g_k)$
6: $\theta_{k+1} = \theta_k + p_k$
7: **end for**

parameters of this neural network. At k-th iteration, full gradient of error function $f(\theta_k)$ is evaluated and (approximated) Hessian matrix is defined as $H(\theta_k)$. Based on this (approximated) Hessian and a proper damping parameter, which aims to make the damped Hessian matrix B_k positive definite and/or avoid B_k being singular. Following this, a quadratic approximation of f around θ_k is constructed as

$$m_k(d) := f(\theta_k) + g_k^T d + \frac{1}{2} d^T B_k d. \tag{2.1}$$

If B_k is positive definite, then we can obtain Newton step d_k by letting $d_k := \arg\min_d m(d) = -B_k^{-1} g_k$. Otherwise, we solve $\min_d m(d)$ by CG method and choose the current iteration whenever a negative curvature direction is encountered, i.e., exist a vector p, such that $p^T B_k p < 0$. If the negative curvature direction is detected at the very first CG iteration, the steepest descent direction $-g_k$ is selected as a descent direction.

Marten [51] modified Algorithm 2.1 in several ways to make it suitable for DNNs. Within neural network, Hessian-vector can be calculated by a forward-backward pass which is roughly twice the cost of a gradient evaluation. On the other side, due to non-convexity

of error function f, Hessian matrix is more likely to be indefinite and therefore a Gauss-Newton approximated Hessian-matrix is used. Note that Gauss-Newton is positive semi-definite matrix but it can be treated as a good approximation only if the current point is close to local minimizer, which motivates our work to design a Hybrid approach. Moreover, pre-conditioning and a CG-backtracking technique is used to decrease the number of CG iterations and obtain best descent direction. However, it is claimed in [98] that such techniques are not very helpful and even tend to make the degrade performance owing to the increased compute and storage requirements. Therefore, we skip these steps and directly move on to our distributed HF algorithm depicted in Algorithm 2.2. For example, to calcu-

Algorithm 2.2 Distributed Hessian-Free Algorithm

1: **Initialization:** θ_0 (initial weights), λ (initial damping parameter), δ_0 (starting point for CG solver), N (number of MPI processes), distributed data
2: **for** $k = 1, 2, \ldots$ **do**
3: Calculate gradient $\nabla f_{[i]}(\theta_k)$ on each node $i = 0, \ldots, N-1$
4: Reduce $\nabla f_{[i]}(\theta_k)$ to root node to obtain full gradient $g_k = \frac{1}{N}\sum_{i=0}^{N-1} \nabla f_{[i]}(\theta_k)$
5: Construct stochastic (approximated) Hessian-vector product operator $G_k(v)$
- Calculate Hessian-vector product $\nabla^2 f_{[i]}(\theta_k)v$ corresponding to one Mini-batch on each node $i = 0, \ldots, N-1$
- Reduce $\nabla^2 f_{[i]}(\theta_k)v$ to root node to obtain $G_k(v) = \frac{1}{N}\sum_{i=0}^{N-1} \nabla^2 f_{[i]}(\theta_k)v$

6: Solve $(G_k + \lambda I)(v) = -g_k$ by CG(BI-CG) method with starting point 0 or $\eta\delta_{k-1}$ (λ is damping and η is decay)
7: Use CG solution s_k or possible negative curvature direction d_k to find the best descent direction δ_k
8: Update λ by Levenberg-Marquardt method (Marten 2010)
9: Find α_k satisfying $f(\theta_k + \alpha_k\delta_k) \leq f(\theta_k) + c\alpha_k g_k^T \delta_k$ (c is a parameter)
10: **Update** $\theta_{k+1} = \theta_k + \alpha_k\delta_k$
11: **end for**

late full gradient (or Hessian vector product needed by CG solver), each node is responsible for computing the gradient (Hessian vector product) based on data samples stored locally. A reduction step is followed to aggregate them to a root node.

2.3.2 Dealing with Negative Curvature

As mentioned in [17], to minimize a non-convex error functions over continuous, high dimensional spaces, one may encounter proliferation of saddle points which are surrounded by high error plateaus. One shortage coming from the use of first-order methods like SGD is that it can not recognize curvature information, and therefore dramatically slow down the learning rate around such saddle points. The saddle-free Newton method (SFN) [17] is then proposed to identify and escape such saddle points. However, they build an exact Hessian

to accomplish SFN on a small size neural network. However, this is impractical or even infeasible for medium or large scaled problems. In this chapter, we propose another method to exploit the local non-convexity of the error function even for a large size network.

A negative curvature direction at current point θ of function f is defined as a vector such that the direction is descent ($g^T d \leq 0$) and also is dominant in the negative eigenspace ($d^T H d < 0$), where g, H are gradient and Hessian of f at point θ.

One naïve way how to find a negative curvature direction is to choose an eigenvector u associated with a negative eigenvalue of H. Then a possible curvature direction is chosen from $\{-u, +u\}$ to ensure that $g^T d \leq 0$. Note that if positive semi-definite, no negative curvature direction can be found. In general, it is computationally expensive to find an eigenvector associated with smallest eigenvalue. Therefore a parameter $\mu \in (0, 1)$ is chosen and a sufficient negative descent direction should satisfy $d^T H d \leq \min(0, \mu \lambda_{min}(H))$, where $\lambda_{min}(H)$ is the smallest eigenvalue of H.

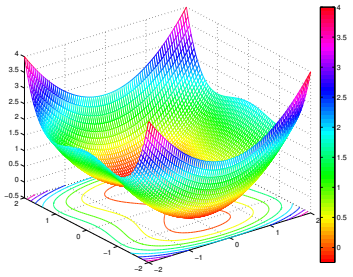


Figure 2.2: A simple 2D example which has one saddle point $(0, 0)$ and two local minimizer $(0, 1)$ and $(0, -1)$.

In other words, we intend to find negative curvature directions, (i.e., direction d such that $d^T H(x) d < 0$). Actually, along with those negative directions, the approximated quadratic model is unbounded below, which shows potential of reduction at such direction (at least locally, while the quadratic approximation is valid). It was shown in [67] that if algorithms uses negative curvature directions, it will eventually converge to second order critical point.

We show a 2D example [61] in Figure 2.2, where the function is $f(x, y) = 0.5x^2 + 0.25y^4 -$

$0.5y^2$. It is easy to obtain that

$$\nabla f = (x, y^3 - y)^T, \text{ and } \nabla^2 f = \begin{bmatrix} 1 & 0 \\ 0 & 3y^2 - 1 \end{bmatrix} \tag{2.2}$$

and therefore three stationary points are obtained. Starting with any initial point of the form $(x, 0)^T$, the (stochastic) gradient descent method will always converge to saddle point $(0, 0)^T$. Actually, even for common second order method (Naïve Newton method [65], Truncated Newton method [65, 51], Saddle Free Newton method [17]), they all converge to saddle point $(0, 0)^T$. The reason is that none of such algorithm can provide a direction along y-axis, which is a negative curvature direction. In this 2D-example, negative curvature direction can be chosen as $d = (0, -1)^T$ (the eigenvector associated to negative eigenvalue -1 of $\nabla^2 f$) at saddle point $(0, 0)^T$ and therefore, we escape saddle point $(0, 0)^T$ and achieve local minimum.

We are now ready to show an improved method to find a possible negative curvature by stabilized bi-conjugate gradient descent (Bi-CG-STAB, Algorithm 2.3), which is a Krylov method that can be used to solve unsymmetrical or indefinite linear system [77]. The benefits of using Bi-CG-STAB is that we can use exact stochastic Hessian information (which may not be positive definite) instead of using Gauss-newton approximation, which will lose the curvature information. It is shown in [51] that HF-CG is unstable and usually fails to convergence. The reason behind that is a fact that HF-CG ignores negative curvature. At the point where the Hessian has relative large amount of negative eigenvalues, it is also inefficient to find a descent direction by restarting the CG solver and modifying the damping parameter.

To use BI-CG-STAB, we set a fixed number of CG iterations [39] and choose the candidates of descent direction for CG-backtracking [51] by letting $\tilde{d} = -\text{sign}(g^T d)d$. Therefore, at each CG iteration, either an inexact CG solution where $\tilde{d}^T H \tilde{d} > 0, g^T \tilde{d} < 0$ is found or an negative curvature direction where $\tilde{d}^T H \tilde{d} < 0, g^T \tilde{d} < 0$ is found.

Algorithm 2.3 Bi-CG-STAB Algorithm

Compute $r_0 := b - Ax_0$. Choose r_0^* such that $(r_0, r_0^*) \neq 0$
$p_0 := r_0$, $k := 0$
if Termination condition not satisfied **then**
    $\alpha_j := (r_j, r_0^*)/(Ap_j, r_0^*)$
    $s_j := r_j - \alpha_j Ap_j$
    $\gamma_j := (s_j, As_j)/(As_j, As_j)$
    $x_{j+1} := x_j + \alpha_j p_j + \gamma_j s_j$
    $r_{j+1} := s_j - \gamma_j As_j$
    $\beta_j := \frac{(r_{j+1}, r_0^*)}{(r_j, r_0^*)} \times \frac{\alpha_j}{\gamma_j}$
    $p_{j+1} := r_{j+1} + \beta_j(p_j - \gamma_j Ap_j)$
end if

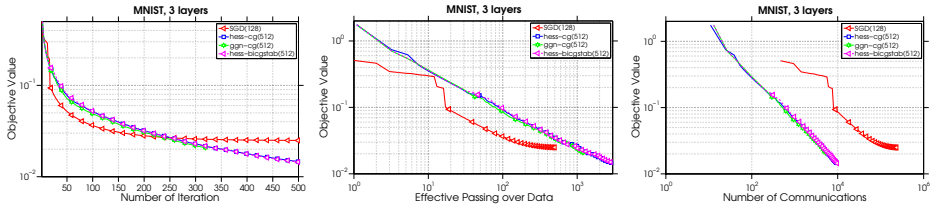


Figure 2.3: Performance comparison among SGD and Hessian-free variants.

2.4 Numerical Experiments

We train MNIST (images) and TIMIT (speech) dataset with various number of hidden layers and hidden units. Note that we do not do any distortions or pre-training for these two dataset as we are interested in scaling and stability of the methods.

2.4.1 Comparison of Distributed SGD and Distributed Hessian-free Variants

In Figure 2.3 we train MNIST dataset with one hidden layers of 400 units, with $N = 16$ MPI processes and compare the performance of four algorithms in terms of the objective value vs. iterations (left), effective passes over data – epochs (middle) and number of communications (right). Note that for presentation purposes we count one epoch of SGD as "one iteration", even-though it is $n/(N \times b)$ iterations. If we look on the evolution of objective value vs. iterations, all algorithms looks very comparable, however, if we check the evolution of objective value vs. epochs, we see that each iteration of second order method requires multiple epochs (one epoch for computing full gradient and possibly many

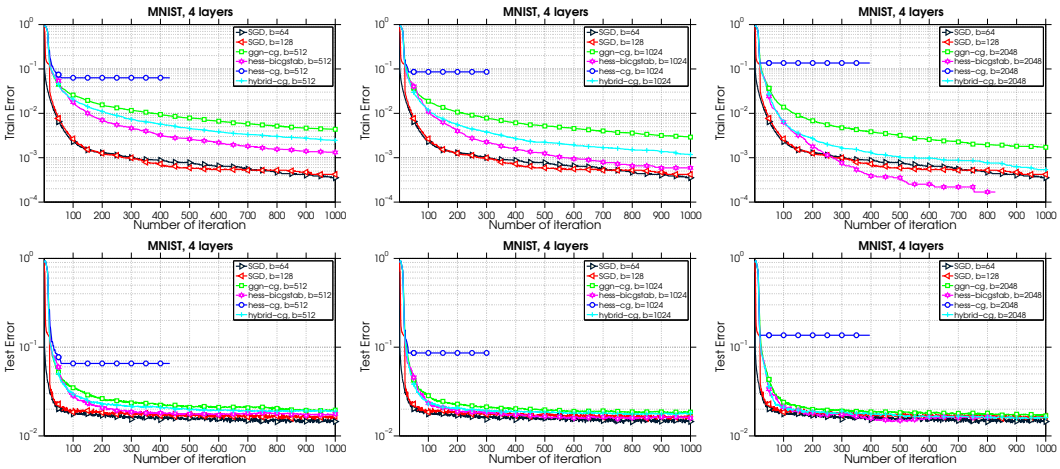


Figure 2.4: Performance comparison among various size of mini-batches on different methods (3 plots above). The neural network has two hidden layers with size 400, 150.

more for a line-search procedure). This can be seen as the trade-off due larger mini-batch sizes, because of which the number of updates within an epoch (one-pass through all the samples) is reduced. We currently looking into methods to address this issue which typical of large-batch second order methods. We would like to stress, that in a contemporary high performance clusters each node is usually massively parallel (e.g. in our case 2.65 Tflops) and communication is usually a bottleneck. The very last plot in Figure 2.3 shows the evolution of objective value with respect to communication. As it is apparent, SGD needs in order of magnitude more communication (for 1 epoch it needs $n/(Nb)$ communications). However, increasing b would decrease number of communications per epoch, but it would significantly decrease the convergence speed. We can also see that SGD got stuck around training error 0.01, whereas second order methods continues to make significant additional progress.

In Figure 2.4 we show how increasing the size of a batch, i.e., from 512 to 2048 is accelerating convergence of second order methods. On contrary, increasing batch size for SGD from $b = 64$ to $b = 128$ (beyond which the SGD-performance largely deteriorates). This also implies that increasing batch size to decrease communication overhead of SGD will slow down the method. Hybrid-CG is a method that uses Hessian information and Gauss-Newton information alternatively. At the beginning, when the starting point may be far away from local minimizer, we use Hessian-CG method and whenever a negative curvature

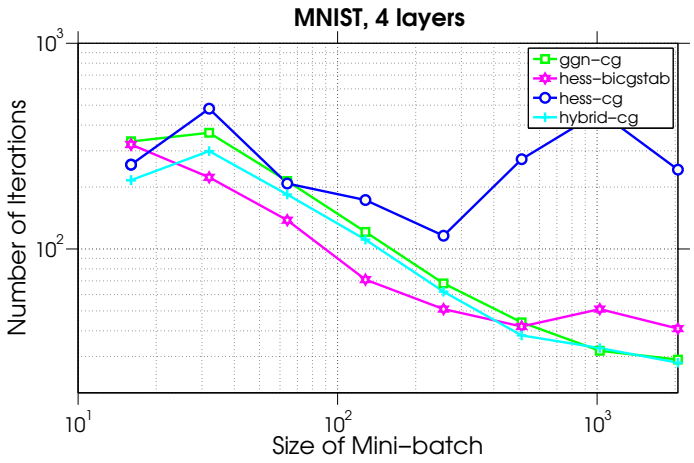


Figure 2.5: Number of iterations required to obtain training error 0.02 as a function of batch size for second order methods.

is encountered, we turn to use Gauss-Newton Hessian approximation for next iteration, and after this iteration, Hessian-CG is used again. The intuition behind it is that we want to use the exact Hessian information as much as possible but also expected to have a valid descent direction at each iteration. From Figure 2.4, we observe that unlike SGD method, Hessian-free variants (except Hessian-CG), are able to make further progress by reducing objective value of error functions, as well as training error continuously. Meanwhile, our proposed Hessian-Bi-CG-STAB outperforms other Hessian-free variants, which shows consistently in all three figures (and others figures in Appendix). If we consider the scaling property in terms of mini-batch, we can see that as the size of mini-batch increase, Hessian-free variants actually *performs better*. The intuition behind it is that larger b is making the stochastic Hessian approximation much closer to the true Hessian. Figure 2.4 right shows scaling of convergence rate as a function of mini-batch. In the plot, b represents the size of mini-batch and the y-axis is the number of iteration the algorithm needed to hit training error 0.04. We see that as we increase the size of mini-batches, it takes less iteration to achieve a training error threshold. The reason is that with a larger mini-batches, we are able to approximate the Hessian more accurate and it is then good to find an aggressive descent direction.

2.4.2 Scaling Properties of Distributed Hessian-free Methods

Let us now study scaling properties of existing and proposed distributed Hessian-free methods. All experiments in this section were done on the large TIMIT speech recognition data-set, with 360 features, 1973 classes, and 1013950 samples. The samples are split into two parts, where we use 70% as training data-set and 30% as testing data-set. The network is set to have 3 fully-connected hidden layers with 512 units each. In Figure 2.6 (top-left) we show the scaling or all studied second order methods with respect to the number of nodes. Each node has two sockets, which correspond to two non-uniform memory (NUMA) regions. To exploit this we run a MPI rank per socket and within the socket we use the multi-threaded Intel MKL functions for the BLAS kernels (sgemm, sgemv), which make up the core compute - to utilize the available 18 cores.

The picture on left shows how the duration of one iteration scale with number of nodes for various size of batch size. Observe, that the scaling is almost linear for values $B \geq 4096$. Actually, the small batch size is the primary bottleneck for scaling because of the limited parallelism. Hence this larger batch-size (increased parallelism) is essential for scaling to larger number of nodes. As was show in 2.4 large batch-size are generally only beneficial for second order methods (as opposed to SGD). Figure 2.6 (top, last 3 plots) shows the speed-up property of the 3 main components of the second order algorithm. Note that both gradient computation and line search inherit similar behavior as the total cost of one iteration. In case of CG, we see that the time of one CG is increasing with increasing size of nodes. The reason for it is that Hessian-vector product is evaluated only for one batch (whose time should be independent from the number of nodes used) but the communication time is naturally increased with mode nodes. It reminds us to remark that the time of communication in this case is comparable to the local compute and hence the pictures suggest very bad scaling. Let us stress that the time of one CG is in order of magnitude smaller then computing of full gradient or line search procedure. As an immediate next step, we are looking into more comprehensive characterization of the compute and bottleneck analysis of both single and multi-node performance. Figure 2.6 (bottom) shows the each batch size the time of 3 major components of the algorithm.

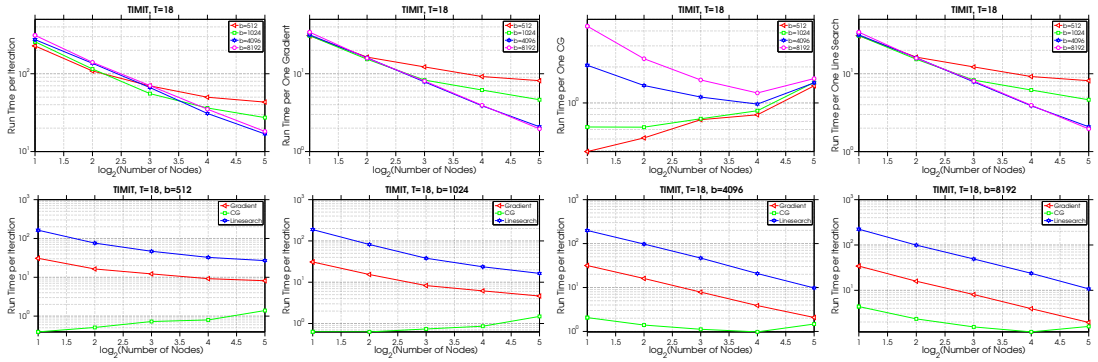


Figure 2.6: Performance scaling of different part in distributed HF on upto 32 nodes (1,152 cores).

2.5 Conclusion

In this chapter, we revisited HF optimization for deep neural network, proposed a distributed variant with analysis. We showed that unlike the parallelism of SGD, which is inherently sequential, and has limitation (large batch-size helps to scale but leads to slows convergence). Moreover, a cheap way to detect curvature information and use negative curvature direction by using BI-CG-STAB method is discussed. It is known that to use of negative curvature direction is essential on improving the training performance. Furthermore, a Hybrid variant is discussed and applied. We show a significant speed-up by applying distributed HF in numerical experiment and the basic comparison among SGD and other HF method shows competitive performance.

Chapter 3

Steps towards Successful Training of Deep Neural Networks Using Second order Optimization Methods

In this Chapter, we explore second order algorithms developed for the very recent useful areas of machine learning, namely deep learning networks. First we discussed the general form of the deep neural network in mathematic formula and some brief discussion of recent work and our work. Then a more detailed introduction in terms of different network architectures and first/second order oracles as the supportive of our algorithms are discussed. The following content discussed mainly on our proposed second order algorithms and observation obtained from our numerical experiments. The conclusion and future word are stated at the end of this Chapter.

3.1 Introduction

Deep neural networks models has been used for achieving state-of-the-art results on a wide variety of tasks including image-classification and objects recognition [20, 43], Natural Language Processing [31], speech recognition [26], etc. In the past few decades, many different neural network architectures have been considered to apply on real-world applications Convolutional Neural Networks (CNNs) for processing data with a known grid-like structure, or

Recurrent Neural Networks (RNNs) for addressing tasks involving time dimension in data. The development of pre-training, better forms of initialization [53], fruitful variants of training techniques and improved hardware have made it possible to train very deep network and achieve excellent performance.

At the core of training deep neural networks exists a complex and highly nonconvex optimization problem. For a muti-label classification problem, given n sample-label pairs $(x_i, y_i)_{i=1}^n$, we construct neural network models h with respect to parameter θ to obtain the predicted labels $\hat{y}_i = h(x_i, \theta)$ for each input sample x_i. If we denote the loss function for the i-th sample by $f(\hat{y}_i, y_i)$, the overall training loss for the entire sample set is then defined by

$$F(X, Y; \theta) = \frac{1}{n} \sum_{i=1}^{n} f(\hat{y}_i, y_i), \tag{3.1}$$

where the loss function $f_i(\theta) \doteq f(\hat{y}_i, y_i)$ may include the squared error $\|\hat{y_i} - y_i\|^2/2$ and the cross-entropy error $-\sum_j (y_{ij} \log(\hat{y_{ij}}) + (1 - y_{ij}) \log(1 - \hat{y_{ij}}))$. Note that all of the loss functions are nonnegative. The ultimate goal is then to minimize the overall training loss (3.1) to obtain the best parameter θ^* such that the least classification error on both the validation and testing datasets is achieved.

Currently, the most popular methodologies to train networks are in the category of first-order (or gradient-based) optimization framework, like mini-batch stochastic gradient method (MSGD), mini-batch stochastic gradient method with momentum (ASGD) [92], and other variants such as Adagrad [22], Adadelta [101], Adam [38], etc. There are also plenty of practical improving techniques to enhance the training performance, such as drop-out [90], batch normalization [33], layer normalization [2], to name but a few.

Challenges. In training neural networks, especially when addressing deep neural networks with a large amount of data samples, one of the main challenges is the relatively slow training rate after some initial success as empirical evidence shown in [17]. Besides, computational results claim that it is more likely to achieve better training/testing performance when the optimization algorithms could help converge to a local minimizer of training loss function defined in (3.1). However, since the models defined by deep neural networks are always

highly non-convex, the number of saddle points increases exponentially as the number of hidden layers and corresponding neurons increases. Within the neighborhood of saddle points, the first-order methods may hardly make progress due to the nearly zero gradient of the loss function. Therefore, the first-order methods suffer to escape from saddle points and show frustrating slow convergence rate after initial progress. Recent work [59] suggests to add noise to the stochastic gradients to prevent slowdown near a saddle point. We will also discuss the noising method later. In this chapter, our focus is on the second order (curvature-based) methods which were also highlighted in [17].

The second order methods, as an alternative to training deep neural network, were widely discussed in recently years. Examples include Hessian-free optimization in [51, 52], L-BFGS optimization in [5] and saddle-free Newton (SFN) method in [17]. The extensions of the original work including the improvement of the preconditioning matrix for conjugate gradient (CG) solver [78], as well as the parallel/distributed variants for second order methods [28]. Among the previous works, either fully connected feed forward neural networks (DNNs) or recurrent neural networks (RNNs) were considered.

Our main objective in this chapter is to explore second order methods and take advantage of the negative curvature information to help escape saddle point efficiently. In thic chapter, Inexact Stochastic Newton-CG method (SINNC) and Inexact Stochastic Trust Region method (SINTR) are proposed. We develop efficient ways to detect negative curvature directions which can be used to overcome the saddle point issue described in [17]. We also record the full Hessian eigenvalues distribution for visualizing the efficiency of our algorithms in terms of escaping the saddle points and compare it with also first-order approaches which will get stuck near saddle points. By visualizing the performance with respect to local minimal convergence, it clearly shows that our proposed second order methods will escape the saddle while the first-order can hardly do this. Around the saddle point where the Hessian contains negative eigenvalues, we are likely to get the negative descent direction and converge to a semi-positive or nearly zero Hessian second order stationary point.

3.1.1 Fully Connect Deep Neural Network

Deep learning networks play an important role among various tools used for the machine learning problems. They could be applied on both supervised learning like image classification or speech classification and also unsupervised learning like auto-encoder for finding the patterns and compressed representations of the data.

The good generalization properties of deep learning networks make it possible to tackle complex problems where it is not efficient to abstract the underlined patterns and feature representations.

We set a common used hand-written digits dataset named MNIST as an example for illustration how the networks performed on the particular dataset. The MNIST dataset consists of the pixel data of 60000 hand-written digits and all digits are range from 0-9. As an small dataset for quick test of neural network performance, the MNIST dataset is used widely.

The fully connected network is constructed by connecting every neuron in the previous layer, and each connection has it's own weight. It's a general purpose of connection pattern where no assumptions among the features are considered in the data.

3.1.2 Deep Convolutional Neural Network

Deep convolutional neural networks [43, 45, 89] are widely used in tasks like image classification, speech recognition, objective detection, etc. Such networks aim to track problems with local similarity, where common feature representations could be shared among different samples and equally likely to occur anywhere. In a convolutional layer, each neuron is only connected to a few nearby neurons in the previous layer. The same set of weights (filter) is used for deriving every neuron.

3.2 Second order Methods for Deep Neural Networks

In this Section, we follow Martens' notation [51] to derive necessary oracles needed for constructing the second order methods, mainly on loss value, gradient, and Hessian-vector multiplication evaluation. Note that we mainly focused on network structure for supervised

learning like classification problem. As introduced in Section 3.1, a neural network could be denoted by a function $f(X;W)$, where X is the input fed into network and W represents the set of weight matrix W_l and bias vector b_l, $\forall l \in [L] := \{1, 2, \ldots, L\}$. L is the total number of layers and l represents a specific layer in the network.

We derive these oracles for fully connected layers (FC), convolutional layers (Conv), and also loss layers(Loss), so that they could be applied on both fully connected network and convolutional neural network. The input is representing as a matrix $X \in \mathbb{R}^{d\times N}$, where d is the number of input features and N is the number of batch size we fed into networks, i.e., N samples are used for evaluating these oracles. Assume we are now at layer l, we are going to derive the needed oracles in the following Section.

3.2.1 First Order Oracle

Loss value evaluation ($l^{th} \to (l+1)^{th}$ layer):

- **FC:** For the fully connected layer, the input is formulated as a matrix $a_l \in \mathbb{R}^{n_l \times N}$, and we further have

$$s_{l+1} = W_l^T a_l + b_l, \tag{3.2}$$

where $s_{l+1} \in \mathbb{R}^{n_{l+1}\times N}$ are the units value before activation of the next layer and $W_l \in \mathbb{R}^{n_l \times n_{l+1}}$ is the connected weights between layer l and $l+1$, and $b_l \in \mathbb{R}^{n_{l+1}\times N}$ is the bias term. Each layer computes the activations as

$$a_{l+1} = \phi_{l+1}(s_{l+1}), \tag{3.3}$$

where ϕ_{l+1} adds non-linearity on each unit of the $(l+1)^{th}$ layer. We denote unction ϕ by the non-linear activation functions. The common used activation function including sigmoid function, tanh function or Relu function and its variants (PRelu, LRelu), as follows:

$$\phi(x) = \begin{cases} \frac{1}{1+e^{-x}}, & \text{sigmoid}, \\ \frac{e^x - e^{-x}}{e^x + e^{-x}}, & \tanh, \\ \max\{0, x\}, & \text{Relu}. \end{cases} \tag{3.4}$$

The $\phi_l : \mathbb{R}^{n_l} \to \mathbb{R}^{n_l}$ is then defined as a vectorized function for applying the activation function ϕ to each unit of l^{th} layer, respective.

- **Conv:** For the convolutional layer, the input is formulated as a 4d tensor $a_l \in \mathbb{R}^{N \times d_l \times h_l \times w_l}$, where $n_l = d_l \times h_l \times w_l$ is the number of units at the l^{th} layer input, also d_l, h_l, w_l are remarked as depth, height, and width of layer input, respectively. The units value of the $(l+1)^{th}$ layer $s_{l+1} \in \mathbb{R}^{N \times d_{l+1} \times h_{l+1} \times w_{l+1}}$ is then computed as

$$(s_{l+1})_{(p,i,j,\hat{p})} = \sum_{\hat{i}=0}^{\hat{I}_l} \sum_{\hat{j}=0}^{\hat{J}_l} \sum_{\hat{k}=0}^{d_l} (W_l)_{(\hat{p},\hat{i},\hat{j},\hat{k})} (a_l)_{(p,i+\hat{i},j+\hat{j},\hat{k})} + (b_l)_{(p,i,j,\hat{p})} \tag{3.5}$$

for $(p, i, j, \hat{p}) \in [N, d_l, h_l, w_l]$, where $W_l \in \mathbb{R}^{d_{l+1}, \hat{I}_l, \hat{J}_l, d_l}$ is the filter weights between layer l and $l+1$, and $b \in \mathbb{R}^{N \times d_{l+1} \times h_{l+1} \times w_{l+1}}$ is the bias term. We also denote this operator as $s_{l+1} = \text{convf}(W_l, a_l) + b_l$ and the operator will be used in second order oracle as well. It follows with

$$a_{l+1} = \phi_l(s_{l+1}). \tag{3.6}$$

for getting the output of $(l+1)^{th}$ layer.

- **Loss:** For the loss layer, the input is formulated as a matrix $a_l \in \mathbb{R}^{n_L \times N}$ where n_L is the number of units of the last layer and y are the expected output provided by original data. Then the loss value of the network is derived as

$$f(X; W) = \ell(a_L, y). \tag{3.7}$$

where ℓ could be chosen as the square loss where

$$\ell(a_l, y) = \frac{1}{2N} \sum_{i=1}^{N} \|(a_l)_i - y_i\|^2$$

or softmax-cross-entropy loss if for probabilistic classification problem where

$$\ell(a_l, y) = \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{n_L} y_{ij} \log(\text{softmax}((a_L)_i)_j)$$

and $\text{softmax}(a) := e^a / \sum_{i=1}^{n_L} e^a, \forall a \in \mathbb{R}^{n_L}$ is the softmax operator to normalize a into

$[0, 1]$.

Note that by following the procedure, one could derive the total loss value. The final loss of the network is average sum of losses for each sample (each batch). Therefore the gradient can be computed by summing the gradient for different sample. Many studies use first order information in order to train the network. The most popular method is named Stochastic Gradient Descent (SGD) method, since the stochastic gradient would be computed separately regarding each sample (each batch). Whenever it comes a new sample, one could easily update the gradient and update the weights in the network by a cheap iteration of SGD.

In practice, the most important methodology to compute the (stochastic) gradient is name back-propagation described as follows. By applying the chain rule at each layer for propagating the error through the network, the gradient in terms of all weights and bias can be obtained.

Gradient evaluation ($(l+1)^{th} \to l^{th}$ layer):

- **Loss:** For the loss layer, the gradient of update is derived based on activations given by the forward propagation. Therefore we have

$$\frac{\partial f}{\partial a_L} = \nabla_{a_L} f(a_L, y), \tag{3.8}$$

  where $a_L \in \mathbb{R}^{n_L \times N}$ is the output the last layer (L^{th} layer).

- **Conv:** For the convolutional layer, the gradient of update is derived as follows, the input is a 4d-tensor $\frac{\partial f}{\partial a_l}$ at layer l, and the output is another 4d-tensor $\frac{\partial f}{\partial a_{l-1}}$ at layer $l-1$. We denote $\circ$ by the operator for computing the elementwise product of two vector. By applying the chain rule, the wanted gradient $\frac{\partial f}{\partial W_l}$ and $\frac{\partial f}{\partial b_l}$ could be derived

as follows:

$$\frac{\partial f}{\partial s_l} = \frac{\partial f}{\partial a_l} \circ \phi'(s_l), \tag{3.9}$$

$$\frac{\partial f}{\partial W_l} = \text{convbw}(\frac{\partial f}{\partial s_l}, a_{l-1}), \tag{3.10}$$

$$\frac{\partial f}{\partial b_l} = \text{convbw}(\frac{\partial f}{\partial s_l}, I_{l-1}), \tag{3.11}$$

$$\frac{\partial f}{\partial a_{l-1}} = \text{convba}(W_l, \frac{\partial f}{\partial s_l}), \tag{3.12}$$

where I_{t-1} is defined as the all one element 4d tensor of the same size as a_{l-1}. And similar to the operator $convf$, the operator $convbw$ for calculating $\frac{\partial f}{\partial W_l}$ is defined as following:

$$(\frac{\partial f}{\partial W_l})_{(p,i,j,k)} = \sum_{\hat{i}=0}^{h_l} \sum_{\hat{j}=0}^{w_l} \sum_{\hat{k}=0}^{d_{l-1}} (\frac{\partial f}{\partial s_l})_{(p,\hat{i},\hat{j},k)} (a_{l-1})_{(p,i+\hat{i},j+\hat{j},\hat{k})}, \tag{3.13}$$

for $(p, i, j, k) \in [d_l, \hat{I}_l, \hat{J}_l, d_{l-1}]$ and the operator $convba$ for calculating $\frac{\partial f}{\partial a_{l-1}}$ is defined as following:

$$(\frac{\partial f}{\partial a_{l-1}})_{(p,i,j,k)} = \sum_{\hat{i}=0}^{\hat{I}} \sum_{\hat{j}=0}^{\hat{J}} \sum_{\hat{k}=0}^{d_l} (W_l)_{(k,\hat{i},\hat{j},\hat{k})} (\frac{\partial f}{\partial s_l})_{(p,i-\hat{i},j-\hat{j},k)}, \tag{3.14}$$

for $(p, i, j, k) \in [N, w_{l-1}, h_{l-1}, d_{l-1}]$.

- **FC:** For the fully connected layer, the input is a matrix $\frac{\partial f}{\partial a_l} \in \mathbb{R}^{n_l \times N}$ and the gradient of update $\frac{\partial f}{\partial W_l}$ and $\frac{\partial f}{\partial b_l}$ are derived as following

$$\frac{\partial f}{\partial s_l} = \frac{\partial f}{\partial a_l} \circ \phi'(s_l) \tag{3.15}$$

$$\frac{\partial f}{\partial W_l} = \frac{\partial f}{\partial s_l} a_{l-1}^T \tag{3.16}$$

$$\frac{\partial f}{\partial b_l} = \frac{\partial f}{\partial s_l} \tag{3.17}$$

$$\frac{\partial f}{\partial a_{l-1}} = W_l \frac{\partial f}{\partial s_l} \tag{3.18}$$

From which, we have prepared all necessary oracles to train neural network by first-order methods.

3.2.2 Second Order Oracle

Beyond the first order oracle, to make use of the power of second order method, one could also derive second order oracle for the networks. Note that to exact represent Hessian matrix (or stochastic Hessian) can be impractical since the huge number of parameters involved when training neural networks. On the other hand, since we could apply numerical algorithms such as Conjugate Gradient (CG) descent as sub-routine in second order optimization framework. It indicates that only the Hessian-vector multiplication is needed to use second order models. The so-called $\mathcal{R}$-operator is then introduced for computing Hessian-vector multiplications [69].

Deriving $\mathcal{R}$-Operator for Second order Information: In order to derive the second order algorithm for training neural network. Pearlmutter[69] observed that Hessian-vector multiplications (denoted by Hv) can be simply viewed as a directional derivative of gradient ∇f with respect to direction (v) as

$$Hv = \lim_{v \to 0} \frac{\nabla_w f(w + rv) - \nabla_w f(w)}{r} = \frac{\partial}{\partial r} f(w + rv)|_{r=0}. \tag{3.19}$$

The $\mathcal{R}$-operator is used to simplify the notation regarding the quantity as

$$\mathcal{R}_v f(w) = \frac{\partial}{\partial r} f(w + rv)|_{r=0}. \tag{3.20}$$

It is then straightforward to show the following properties of this $\mathcal{R}$-operator

$$\mathcal{R}_v(af(w) + bg(w)) = a\mathcal{R}_v f(w) + b\mathcal{R}_v g(w), \tag{3.21}$$

$$\mathcal{R}_v(f(w)g(w)) = \mathcal{R}_v f(w) \cdot g(w) + \mathcal{R}_v g(w) \cdot f(w), \tag{3.22}$$

$$\mathcal{R}_v(f(g(w))) = f'(g(w))\mathcal{R}_v g(w), \tag{3.23}$$

$$\mathcal{R}_v w = v. \tag{3.24}$$

Note that $Hv = \nabla^2 f(w)v = \mathcal{R}_v \nabla f(w)$ and the properties listed above, one could then obtain Hessian-vector multiplication by applying the $\mathcal{R}$-operator to the gradient compu-

tation methods. That's to say, one could apply the first order oracle in Section 3.2.1 to further derive the second order oracle. We also discussed those second order oracles for different type of layers. Note that at the 0^{th} layer $a_0 = X$ is constant function, therefore we have $\mathcal{R}a_0 = 0$. The input vector v is represented by $(\mathcal{R}W_l, \mathcal{R}b_l)_{l=1}^L$, and the Hessian-vector multiplication Hv is represented by $(\mathcal{R}\frac{\partial f}{\partial W_l}, \mathcal{R}\frac{\partial f}{\partial b_l})_{l=1}^L$.

$\mathcal{R}$-value evaluation ($l^{th} \to (l+1)^{th}$ layer):

- **FC:** For the fully connected layer, the input is formulated as a matrix $a_l \in \mathbb{R}^{n_l \times N}$, and we further have

$$\mathcal{R}s_{l+1} = \mathcal{R}W_l^T a_l + W_l^T \mathcal{R}a_l + \mathcal{R}b_l, \tag{3.25}$$

  where $\mathcal{R}s_{l+1} \in \mathbb{R}^{n_{l+1} \times N}$ is the $\mathcal{R}$-unit value before activation of the next layer and $W_l \in \mathbb{R}^{n_{l+1} \times n_l}$ is the connected weights between layer l and $l+1$, $\mathcal{R}W_l \in \mathbb{R}^{n_{l+1} \times n_i}$ and $\mathcal{R}b_l \in \mathbb{R}^{n_{l+1} \times N}$ construct the vector v between l^{th} layer and $(l+1)^{th}$ layer. The activations as following

$$\mathcal{R}a_{l+1} = \mathcal{R}s_{l+1} \cdot \phi_l'(s_{l+1}). \tag{3.26}$$

- **Conv:** For the convolutional layer, the input is formulated as a 4d tensor $a_l \in \mathbb{R}^{N \times d_l \times h_l \times w_l}$, where $n_l = d_l \times h_l \times w_l$ is the number of units at the l^{th} layer input, also d_l, h_l, w_l are remarked as depth, height, and width of layer input, respectively. The units value of the $(l+1)^{th}$ layer $s_{l+1} \in \mathbb{R}^{N \times d_{l+1} \times h_{l+1} \times w_{l+1}}$ is then computed as

$$\mathcal{R}s_{l+1} = \text{convf}(\mathcal{R}W_l, a_l) + \text{convf}(W_l, \mathcal{R}a_l) + \mathcal{R}b_l, \tag{3.27}$$

  where $s_{l+1} \in \mathbb{R}^{N \times d_{l+1} \times h_{l+1} \times w_{l+1}}$ is the unit value before activation of the next layer, $W_l \in \mathbb{R}^{d_{l+1}, \hat{I}, \hat{J}, d_l}$ is the filter weights between layer l and $l+1$, and $b \in \mathbb{R}^{N \times d_{l+1} \times h_{l+1} \times w_{l+1}}$ is the bias term.

  It follows with

$$\mathcal{R}a_{l+1} = \mathcal{R}s_{l+1} \cdot \phi_l'(s_{l+1}). \tag{3.28}$$

- **Loss:** For the loss layer, the input is formulated as a matrix $a_l \in \mathbb{R}^{n_L \times N}$ where n_L is the number of neuron of the last layer. Note that we also have the loss value of the

network

$$f(X;W) = \ell_l(a_l, y), \tag{3.29}$$

where ℓ_l is the vectorized loss functions defined the same as previous section.

In practice, the most important methodology to compute the (stochastic) gradient is name back-propagation described in the following algorithm. The algorithm implements the chain rule at each layer in order to propagate the error through the network, it also can be referred as automatic differential.

Loss: For the loss layer, the gradient of update is derived based on activations give by the forward propagation. Therefore we have $\mathcal{R}\frac{\partial f}{\partial a_L} \in \mathbb{R}^{n_L \times N}$ and

$$\mathcal{R}\frac{\partial f}{\partial a_L} = (\nabla^2_{a_L} f) \circ \phi'(s_L) + W_L \cdot \frac{\partial f}{\partial a_L} \circ \phi''(s_L) \circ s_L. \tag{3.30}$$

Conv: For the convolutional layer, the gradient of update is derived based on the known $\mathcal{R}\frac{\partial f}{\partial a_l}$ as follows

$$\mathcal{R}\frac{\partial f}{\partial s_l} = \mathcal{R}\frac{\partial f}{\partial a_l} \circ \phi'(s_l) + W_l \cdot \frac{\partial f}{\partial a_l} \circ \phi''(s_l) \circ \mathcal{R}s_L, \tag{3.31}$$

$$\mathcal{R}\frac{\partial f}{\partial W_l} = \text{convbw}(\mathcal{R}\frac{\partial f}{\partial s_l}, a_{l-1}) + \text{convbw}(\frac{\partial f}{\partial s_l}, \mathcal{R}a_{l-1}) \tag{3.32}$$

$$\mathcal{R}\frac{\partial f}{\partial b_l} = \text{convbw}(\mathcal{R}\frac{\partial f}{\partial s_l}, I_{l-1}) \tag{3.33}$$

$$\mathcal{R}\frac{\partial f}{\partial a_{l-1}} = \text{convba}(W_i, \mathcal{R}\frac{\partial f}{\partial s_l}) + \text{convba}(\mathcal{R}W_l, \frac{\partial f}{\partial s_l}) \tag{3.34}$$

FC: For the fully connected layer, the gradient of update is derived based on given $\mathcal{R}\frac{\partial f}{\partial a_L}$ as following

$$\mathcal{R}\frac{\partial f}{\partial s_l} = \mathcal{R}\frac{\partial f}{\partial a_l} \circ \phi'(s_l) + W_l \cdot \frac{\partial f}{\partial a_l} \circ \phi''(s_l) \circ \mathcal{R}s_L, \tag{3.35}$$

$$\mathcal{R}\frac{\partial f}{\partial W_l} = \mathcal{R}\frac{\partial f}{\partial s_l} a_{l-1}^T + \frac{\partial f}{\partial s_l} \cdot \mathcal{R}a_{l-1}^T \tag{3.36}$$

$$\mathcal{R}\frac{\partial f}{\partial b_l} = \mathcal{R}\frac{\partial f}{\partial s_l} I_{l-1}^T \tag{3.37}$$

$$\mathcal{R}\frac{\partial f}{\partial a_{l-1}} = W_i \cdot \mathcal{R}\frac{\partial f}{\partial s_l} + \mathcal{R}W_l \cdot \frac{\partial f}{\partial s_l}. \tag{3.38}$$

By continuously following those operators, one could derive the Hessian-vector multiplication represented by $(\mathcal{R}\frac{\partial f}{\partial W_l}, \mathcal{R}\frac{\partial f}{\partial b_l})_{l=1}^{L}$. We now have prepared all necessary oracles to train neural network by second order methods.

3.2.3 Algorithms for Training Neural Networks

In this section, we briefly review the algorithms for training the network using second order information. As we know in [65], the most common second order algorithms for solving the unconstrained optimization problems is try to construct a reasonable second order approximation model of the original function and find the optimal point of the approximated model. With respect of a minimization problem $\min_{x\in\mathbb{R}^n} f(x)$, at iterates x_k, a quadratic model $m_k(d)$ is built as

$$m_k(d) = f(x_k) + \nabla f(x_k)^T d + \frac{1}{2} d_k^T B_k d_k, \tag{3.39}$$

where B_k is the approximated Hessian information at current point x_k. Depending on how accurate the B_k is approximated to true Hessian, one could solve the problem for achieve a good iterates for the next step. Considering that under neural works setting, it's impractical to derive the exact Hessian H_k or approximated Hessian B_k exactly. Therefore, the use of conjugate gradient methods are much more popular among researchers, since in conjugate gradient methods, the full representation of the matrix B_k is not required and only matrix-vector multiplication is needed (derived in Section 3.2.2).

One issue that arise from the using of conjugate gradient method is the requirement of the positive definiteness of the Hessian approximation, otherwise, the algorithm would be broken down unexpectedly. In order to overcome the issue, Martens' [51] proposed the use of positive semi-definite approximation of the Hessian for the training process. The generalized Gauss Newton (GN) matrix is used in his work. The GN matrix-vector multiplication could be computed by applying the $\mathcal{R}$-operator along with the back-propagation as well. However, one other issue comes out that since GN is only the semi-positive definite approximation to the actually Hessian, therefore, it may stop at a saddle point rather than a local minimum. In other words, as it ignores the negative curvature direction of a non-convex problem, a

good convergence guarantee of local optimal is not obtained. Therefore, we aim to purpose algorithm which would still use conjugate gradient method while able to explore negative curvature information for further reduce the sub-optimality.

3.2.4 Saddle-points Issue on Training Neural Networks

In the paper [17], the saddle point issue is fully discussed in the field of training neural networks. Experimental results claims that by escaping saddle points and achieve a local minima of the train loss, one could achieve better performance of the networks. As stated in [12], these local optimum could achieve the equal performance as global optimum, which stress the importance of finding a local minimum and throw the request of escaping saddle point. As stated in [17], the number of saddle point increases exponentially along with the number of parameters. All the above illustrate that it's important to escape saddle point and achieve the local minimum for better training performance of neural network. Rong Ge els' [24] proposed the type of strictly saddle function and proved that the stochastic gradient descent methods with noise would converge to local minimum in polynomial time almost surely. Others work include the use of cubic regularization [62] which involves a lower bounded third-order subproblem, the saddle-free method proposed by [17] that use the absolute Hessian and trust region framework.

3.2.5 Almost Sure Convergence to a Local Minimizer

To the best of our knowledge, there haven't been a close and full understanding of neural network in the field of optimization. However, empirically, as discussed in [17], one key point to improve neural network performance could be to escape numerous saddle points. That is to say, one should look for an at least second order stationary point to improve the training performance, where a second order stationary point is defined such that its gradient of the objective function is zero and its Hessian is positive semi-definite. Recent work that tries to achieve this include the, SFN (saddle-free Newton) method [17] and MSGD with added noise [59], etc. It is known that, gradient-based algorithms are usually much sensitive to the curvature condition [65] and therefore a proper learning rate is always required to avoid dramatic oscillations and/or slow converge rate for those gradient-based algorithms.

However, the second order algorithms provide a much more powerful and elegant solution to handle the curvature issue. Within the selective rescaling of gradient alone different curvature direction, not only could we guarantee to find a descent direction, also we could use line-search to expect sufficient reduction at each iteration.

We are now showing how second order methods could help us convergence to a local minimizer almost surely. Several assumptions are claimed in the following subsection serving for Theorem 3.2.5.

Assumption 3.2.1. *The function F is twice continuously differentiable and bounded below on the level set.*

Assumption 3.2.2 (Lipschitz continuity)**.** *For any subset $S \subset [n]$, there exists a constant M_S depending on the size of S, such that for any feasible θ and $\hat{\theta}$, we have $\|H_S(\theta) - H_S(\hat{\theta})\|_2 \leq M_S\|\theta - \hat{\theta}\|_2$.*

Proposition 3.2.3 (Bounded Hessian)**.** *For any $i \in [n]$, and $\theta \in \mathcal{C}$, if both Assumption 3.2.1 and Assumption 3.2.2 hold, we have that $\|\nabla^2 f_i(\theta)\|$ is uniformly upper bounded by a constant K, i.e., $\max_{i\in[n]} \|\nabla^2 f_i(\theta)\|_2 \leq K$.*

Lemma 3.2.4. *Assume that at each iteration t, $S_t \subset [n]$ is sampled independently and sufficiently large. For a fixed accuracy threshold ϵ, starting from 0, the direction d_t generated by SteihaugCG [1] solver will share at least the same reduction as the Cauchy point for the approximate model $m_t(d)$.*

We are now showing that there exists a $c_1 \in (0, 1]$, such that

$$m_t(0) - m_t(d) \geq c_1(m_t(0) - m_t(d^C)).$$

Theorem 3.2.5. *Assume that at each iteration t, $S_t \subset [n]$ is sampled independently and sufficiently large such that Assumptions 3.2.2 holds, which indicates $\theta^* \in \mathcal{C}$. The sequence generated by Algorithm 3.2 (the Algorithm is described in Section 3.4) will converge to the first order stationary point.*

[1] The detailed SteihaugCG will be explained in Section 3.4.

Proof. At t-th iteration, we denote d^C by the Cauchy-point of the approximated model, where

$$d^C = \arg \min_{d \in \text{span}\, \nabla F, \|d\| \leq \Delta} m_t(d).$$

From Lemma 4.5 [65], we know that the reduction of the approximation model at Cauchy point is bounded by

$$m_t(0) - m_t(d^C) \geq \frac{1}{2}\|g_t\| \min(r_t, \frac{\|g_t\|}{\|H_{S_t}\|}) \geq \frac{1}{2}\|g_t\| \min(r_t, \frac{\|g_t\|}{K}).$$

The first-order stationary point convergence is then given by Theorem 4.9 of [65] since there are more model reduction by the early terminated SteihaugCG solver solution than the reduction obtained from Cauchy-point. □

Convergence to a local minimizer. In [46] it was shown that gradient descent, when applied to minimization of any $h \in \mathcal{C}^2$, will converge to strict saddle point with probability zero.[2] However, if we have a non-convex quadratic function and use exact Newton method, we can converge to a saddle point just in one step. To utilize their results in our setting one has to realize that both algorithms presented in this chapter are using just a sub-sampled Hessian and also the Newton/Trust-region step is computed inexactly. Following the proofs in [46]. One see that if we start the algorithm from a random initial point, there is a zero probability that we would end-up in a point which lies in the subspace of a saddle point which corresponds to only positive eigenvalues. Moreover, since function (3.1) is more complex than a simple quadratic function and hence it is even less likely that we should be sent to a point which would be attracted and not escape a strict saddle point (actually due to a random choice of sub-sampled Hessian we can see that we are basically randomly perturbing the solution and hence almost surely it will not end up in the lower-dimensional positive eigen-space).

[2] $\hat{w}$ is a strict saddle point if $\hat{w}$ is a stationary point $\|\nabla h(\hat{w})\| = 0$ and $\lambda_{\min}(\nabla^2 h(\hat{w})) < 0$.

3.3 Inexact Stochastic Newton CG Method (SINNC)

We first state our SINNC algorithm as in Algorithm 3.1. At each iteration, the full gradient is computed and used for finding an inexact stationary point corresponding to stochastic Hessian. We force the direction to be descent by flipping its sign if necessary. The Amijo line search is then followed to ensure sufficient reduction of the loss function at each iteration. Note that unlike the Martens' original method [51], which is actually truncated Newton-CG method [65], we do consider negative curvature information indicated from the stochastic Hessian matrix. And it's also different from the saddle-free Newton (SFN) method proposed in [17]. Our proposed method unitize the stochastic Hessian-vector product but there is no need to evaluate the full Hessian, which is required by SFN methods.

Algorithm 3.1 Inexact Stochastic Newton-CG Method (SINNC)

1: **Input:** Sample and label pairs $(x_i, y_i)_{i=1}^n$, an initial iterate θ_0, an initial CG starter d_0 and CG iteration limit k_{max}, constant $c \in (0, 1)$, sample size $\beta \in [n]$.
2: **for** $t = 0, 1, 2, \ldots$ **do**
3: Evaluate full gradient $g_t = \nabla F(\theta_t)$.
4: Generate batch $S_t \in [n]$ randomly so that $|S_t| = \beta$.
5: Apply the matrix-free CG solver to obtain an inexact solution d_t of the possible indefinite linear system

$$H_{S_t} d = -g_t.$$

6: Decide descent direction as

$$p_t = -\text{sgn}(g_t^T d_t) d_t,$$

where $\text{sgn}(x) = 1, \text{ if } x \geq 0, \text{sgn}(x) = -1 \text{ if } x < 0$.
7: Choose learning rate η_t as the largest element in the set $\{1, c, c^2, ...\}$ such that

$$F(\theta_t + \eta_t p_t) \leq F(\theta_t) + c\eta_t g_t^T p_t.$$

8: Update $\theta_{t+1} = \theta_t + \eta_t p_t$.
9: **end for**

3.3.1 Early Terminated CG for Indefinite System

To train neural network by second order methods, the stochastic Hessian matrix and stochastic general Gaussian-Newton matrix are adopted as the approximation of the Hessian matrix [51], and further build the stochastic quadratic approximated model depending on them.

Since training a deep neural network always involves a very large number of parameters, the exact solution of minimizing the quadratic approximation is prohibitive. Instead, we try to achieve a reasonable inexact solution in a computationally cost effective manner. Since the conjugate gradient method (CG) is often used to achieve an increasingly accurate solution after several iterations, we have decided to apply CG to minimize our quadratic model.

A known deficiency of the CG method is that it becomes unstable when an indefinite Hessian matrix is encountered during the minimization of the quadratic model. The reason behind is that with an indefinite Hessian matrix, we may not find a conjugate direction. Several strategies have been proposed to deal that deficiency [65], such as to modify the indefinite Hessian matrix so that the matrix can be positive and apply the CG solver afterward, or to apply a trust region approach which can always find a descent direction, or to use truncated Newton method, which terminates CG iteration whenever the negative curvature is encountered. In this chapter, we applied an early-terminated CG solver in order to find an inexact solution for the quadratic model. With a good initial point, one could build a sequence of conjugate directions (See definition 3.3.1). From which, we could guarantee to reduce the residue of the system until the terminated condition is satisfied.

Definition 3.3.1. *Let $A \in \mathbb{R}^{n\times n}$ be symmetric and nonsingular. We say that the vectors $u, v \in \mathbb{R}^n \backslash \{0\}$, $u \neq v$ are H_S-conjugate if $u^T A v = 0$, $u^T A u \neq 0$ and $v^T A v \neq 0$.*

We are going to prove that within this early terminated CG method, we could get an approximated solution even for indefinite matrix and the proof is left at Appendix.

Lemma 3.3.2. *Given the stochastic matrix $H_S \in \mathbb{R}^{n\times n}$ is symmetric and nonsingular, for any $x_0 \in \mathbb{R}^n$ such that $p_0 = H_S x_0 + g$ and $p_0^T H_S p_0 \neq 0$, let $\{p_i\}_{i=0}^{n-1}$ be a sequence of nonzero H_S-conjugate directions. Denote $P_k = (p_0, \ldots, p_{k-1})$. Then the sequence x_k generated according to the rule*

$$x_{k+1} = x_k + \alpha_k p_k \text{ and } \alpha_k = -\frac{(H_S x_k + g)^T p_k}{p_k^T H_S p_k}$$

can be written as $x_k = x_0 + P_k y$, where y is the unique solution of

$$P_k^T H_S P_k y - P_k^T p_0 = 0.$$

Denote x^* by the solution of $H_S x + g = 0$, since we assume H_S is nonsingular and then $x^* = -H_S^T g$. Further we could show that the residue of $\|x^* - x_0\|_{H_S}$ decrease monotonically, which indicates that we obtain more accurate solution along with the early terminated CG solver.

3.4 Inexact Stochastic Trust Region Method

In this Section, we propose an inexact stochastic trust region method using stochastic Hessian-vector product and SteihaugCG solver to help escape saddle point.Trust region methods are commonly used to enforce global convergence to such nonconvex optimization problems. It relies on solving a bounded quadratic minimization problem $m_t(d)$ at each iterate θ_t, which is constructed by using the approximated Hessian information. The bound Ω of the quadratic model is chosen so that $m_t(d)$ remains a reasonable approximation of F for any $d \in \Omega$. Usually, it is a hard problem to find the exact solution of the quadratic model, therefore, we rely on SteihaugCG which is introduced by [91] to obtain a reasonable inexact solution.

SteihaugCG is a powerful CG variant to resolve the indefinite subproblem issue [65]. Followed by result of [100], we are able to show that SteihaugCG with stochastic Hessian approximation (which is probably indefinite), would give as least the same reduction as Cauchy point (see Theorem 3.2.5). There are also other variants that aim to use trust region method for training deep neural network [97]. Note that the SFN method proposed by [17] need to evaluate the full Hessian H exactly to accomplish eigenvalue/eigenvector decomposition in order to build absolute Hessian $|H|$ in their trust region based framework. Our approach therefore saves much computation and it would be practical to apply in the real training.

3.4.1 Steihaug Conjugate Gradient Descent Method

Algorithm 3.2 Inexact Stochastic Trust Region Method (SINTR)

1: **Input:** Sample and label pairs: $(x_i, y_i)_{i=1}^n$, initial parameter θ_0, initial trust region radius $r_0 \in (0, R)$. Given constants $\eta_0, \eta_1, \gamma_1, \gamma_2$, and ϵ, where $0 \leq \eta_0 < \eta_1 < 1$, $0 < \gamma_1 < 1 < \gamma_2$, $\epsilon > 0$. Sample size $\beta \in [n]$.

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

3: Evaluate full gradient $g_t = -\nabla F(\theta_t)$.

4: Generate batch $S_t \in [n]$ randomly so that $|S_t| = \beta$.

5: Build an approximation of $F(\theta)$ at θ_t, using stochastic Hessian H_{S_t}

$$m_t(d) := F(\theta_t) + g_t^T d + \frac{1}{2} d^T H_{S_t} d.$$

6: Apply early terminated Steihaug CG solver to obtain an inexact minimizer of $m_t(d), i.e.,$

$$d_t = SteihaugCG(H_{S_t}, g_t, r_t, \epsilon),$$

Note that in this case, d_t is always a descent direction.

7: Set $\rho_t = \frac{F(\theta_t) - F(\theta_t + d_t)}{m_t(0) - m_t(d_t)}$,

$$\begin{cases} \theta_{t+1} = \theta_t + d_t, r_{t+1} = \min\{\gamma_2 r_t, R\}, & \rho_t > \eta_1, \\ \theta_{t+1} = \theta_t + d_t, r_{t+1} = r_t, & \rho_t \in [\eta_0, \eta_1], \\ \theta_{t+1} = \theta_t, r_{t+1} = \gamma_1 r_t, & \text{Otherwise.} \end{cases}$$

8: **end for**

3.4.2 Accelerated SINTR with Adding Momentum

To further explore the advantages of second order methods, we are going to propose a novel momentum for improving the escaping efficiency. Note that in our proposed Algorithm 3.1, although we are able to escape the saddle point, it usually takes many iterations to accomplish this. The algorithm described at Algorithm 3.4 is to reduce the iterates need for escaping. The difference between Algorithm 3.4 and Algorithm 3.2 is that we are making the SINTR+ moves as far as possible from the starting point.

We achieve this heuristics in two steps. 1) As long as we derived the descent direction d_t from Algorithm 3.2, instead of using it directly with step-size equal to 1, an extra line search is followed. Since we notice that around the saddle, the objective value always changes very tiny, we therefore remove the sufficient reduction requirement for convergence guarantee and aim to choose the largest step-size along d_t. 2) After we achieve the furthest move along the descent direction d_t, we also add extra momentum for further performance improving. The momentum is accumulated from the previous direction. This actually helps since we notice

Algorithm 3.3 Steihaug CG Solver for Possible Indefinite System: $d = SteihaugCG(B_t, g_t, \Delta)$

Input: Initial point $d_0 = 0$, and let $p_0 = r_0 = -g_t - B_t d_0 = -g_t$, radius Δ, max cg iterations k_{max}
if $\|r_0\| \leq \epsilon$ **then**
  return: $d = d_0$
end if
for $k = 0, 1, 2, \ldots$ **do**
  if $p_k^T B_t p_k \leq 0$ **then**
    break
  end if
  $\alpha_k = \frac{r_k^T r_k}{p_k^T B_t p_k}$
  $d_{k+1} = d_k + \alpha_k p_k$
  if $\|d_{k+1}\| \geq \Delta$ **then**
    break
  end if
  $r_{k+1} = r_k - \alpha_k B_t p_k$
  if $\|r_{k+1}\| \leq \epsilon \|r_0\|$ **then**
    break
  end if
  $\beta_k = \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}$
  $p_{k+1} = r_{k+1} + \beta_k p_k$
  if $k = k_{max}$ **then**
    break
  end if
end for
Find τ such that $d = d_k + \tau p_k$ and $\|d\| = \Delta$
return: d

that near the saddle, the angles between any two adjacent iterates are very tiny in some iterations (See Figure 3.1). It then motivated us to try future move along the momentum direction ν_t. As stated in 1), as long as we could verify that ν_t is a descent direction, we find the largest step-size for the momentum direction. The update for current iterate is then defined as the sum of descent direction d_t movement and extra momentum descent direction ν_t. As shown in Figure 1, the reduction in the objective function by using SINTR+, is achieved when there is substantial increase in the angle between consecutive iterations. In contrast SINTR cannot sufficiently decrease the objective since the angles of consecutive iterations are always small and do not fluctuate enough.

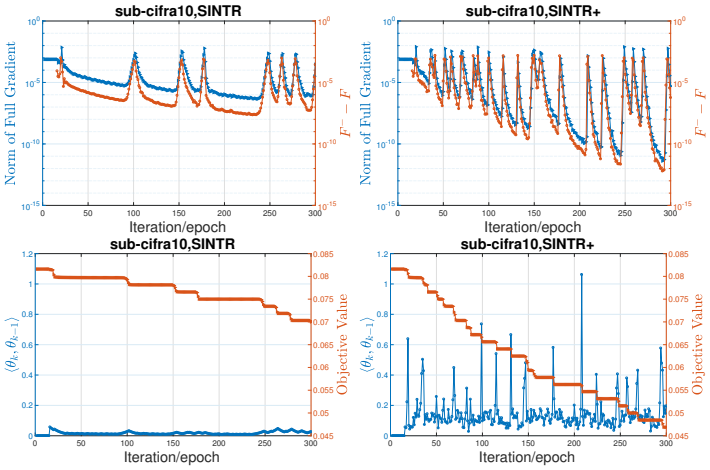


Figure 3.1: Evolution of angles between two adjacent iterative points (bottom row) and the corresponding optimization performance of SINTR and SINTR+.

Algorithm 3.4 Accelerated SINTR with Adding Momentum (SINTR+)

1: **Input:** Initial momentum parameter $\nu_0 = -\mathbf{e}$, constant $0 < c_1 < c_2 < \infty$. Descent direction d_t derived from Algorithm 3.2. Momentum parameter $c \in (0, 1)$.

2: Choose learning rate $\eta_t \in (c_1, c_2)$ as the largest element such that

$$F(\theta_t + \eta_t d_t) \leq F(\theta_t).$$

3: Update $\hat{\theta}_{t+1} = \theta_t + \eta_t d_t$.

4: Set $\nu_t = c\nu_{t-1} + \eta_t d_t$ and flip its sign so that ν_t is a descent direction at $\hat{\theta}_{t+1}$.

5: Choose momentum parameter $\gamma_t \in (c_1, c_2)$ as the largest element such that

$$F(\hat{\theta}_{t+1} + \gamma_t \nu_t) \leq F(\hat{\theta}_{t+1}).$$

6: Update $\theta_{t+1} = \hat{\theta}_{t+1} + \gamma_t \nu_t$.

3.5 Numerical Results

We show our numerical experiments and results discussion in this Section . Our purpose is to illustrate the performance of different algorithms in terms of escaping saddle points and show consistent evidence why we believe that the second order methods will success in the field of training deep neural networks.

We will track the evolution of the full Hessian at each iterates of various algorithms. By doing this, we are able to show the curvature information evolution of the training loss function. Note that it is highly computational demanding to obtain the exact full Hessian in general, letting alone to identify the exact eigenvalues distribution of the Hessian. Therefore,

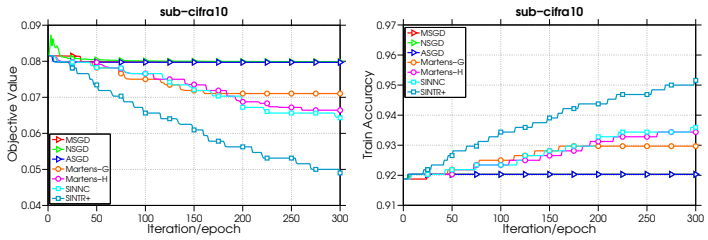


Figure 3.2: Objective value and Training error evolution on various methods on sub-cifra10 dataset starting with a nearly saddle point (gradient norm close to 10^{-6}). Within the first-order methods like MSGD, ASGD and NSGD, there are not much difference among them in terms of escaping saddles and they are all worse than the second order methods. SINTR+ performance the best.

we down sample MNIST and CIFRA10 dataset to have a reasonable number of parameters for our evaluation. We also do experiments on the full MNIST, CIFRA10 to show the superior performance of our purposed algorithms. The dataset set description and network setting (FC1, FC2, FC3) are shown at Table 3.1. The detailed configuration can be found in Appendix.

Dataset	#Features	#Classes	#Samples	Networks
sub-MNIST	100	2	640	FC1
sub-CIFRA10	1024	10	6400	FC2
MNIST	784	10	60000	FC3
CIFRA10	3072	10	50000	FC3

Table 3.1: A list of datasets used in numerical experiments. The prefix sub stands for a down-sampled version of corresponding dataset. All dataset are fitted into the classification model.

3.5.1 Comparison Results Among Various Escaping Approachs

We highlight the performance of common used training methods including MSGD, ASGD[92], NSGD[59], Martens-H[51], Martens-G[51], and also our proposed SINNC, SINTR and SINTR+ methods. All algorithms are fine-tuned to show their best performance. In this experiment, we choose the starting point such that its norm of full gradient ∇F is very tiny (around 10^{-6}), which indicates that it is very close to a saddle point. We concern the evolution of loss objective function value F, norm of full gradient ∇F, training accuracy Acc on all the algorithms and show comparison of different escaping performance in Figure 3.2.

It's claimed in [24] that for a strict saddle function, which is defined such that at any

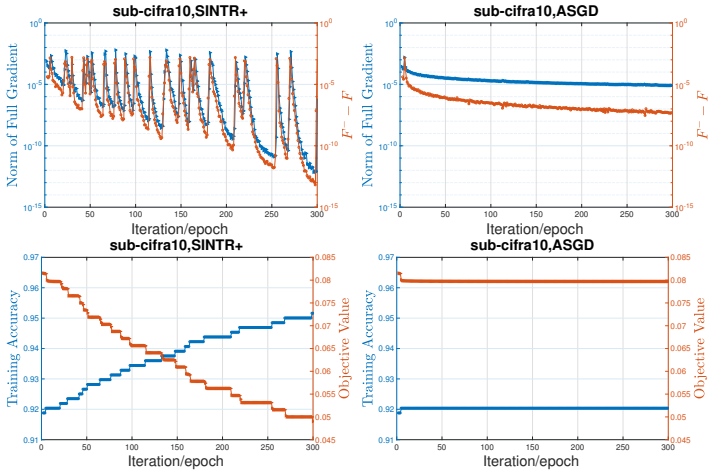


Figure 3.3: Objective value, Gradient norm and Training error evolution on SINTR+ and ASGD in first 300 iterations

stationary point that is not a local minimizer, there must be at least one negative eigenvalue at the point, the naive MSGD method with a suitable noise will almost surely converge to a local minimizer. While in practical, it would be hard to find a suitable noise [59] to achieve the best performance. As one could see from the plot, at nearly saddle point, there is not much progress made by the NSGD approach. It is clearly to show that eventually, our proposed approaches perform much better in terms of escaping.

In Figure 3.3, the left column is the performance of SINTR+ (Algorithm 3.4) and the right row is the performance of ASGD. One could clearly identify how the escaping happens during the training process. Within SINTR+ method, at roughly every 10 iterations, the norm of gradient with go down hill to quit small (gradually decrease) and suddenly, the norm of gradient will jump up together with a decrease of loss objective value and an increase of training accuracy. While when looks at the ASGD algorithm, after 20 iterations, where the first escaping happens, the norm of gradient decrease quit slow and therefore shows a very small progress with respect to the objective value and training error. One could clearly identify that, using our proposed algorithm, we achieve the loss error to 0.05 and training accuracy to 95.2%, while using ASGD, we only achieve loss error as 0.079 and training accuracy to 92.0%, which is almost the same performance as our starting point.

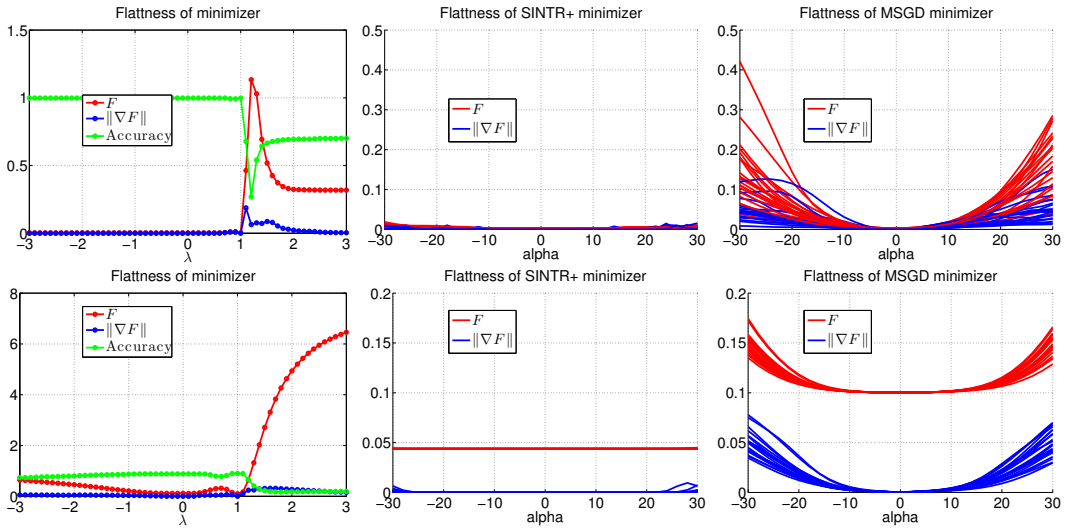


Figure 3.4: Second order methods will converge to flatter minimizer. The first row is the result for CIFRA10 and the second is result of MNIST.

3.5.2 Generalization Gap and Sharp Minima

In this experiment, we show flat minimizer evidence for second order methods (see Figure 3.4). The definition of flat minimizer and model generation is defined from [37]. Starting from a random point, we run MSGD and SINTR+ for a long time to achieve their stable saddle point, denoted by θ_{MSGD} and $\theta_{\text{SINTR+}}$, respectively. In the first figure, we let $\theta(\lambda) = \lambda\theta_{\text{MSGD}} + (1-\lambda)\theta_{\text{SINTR+}}$ and compute $(F, \nabla F, Acc)$ with respect to each $\theta(\lambda)$. Note that when $\lambda = 0$, we are at SINTR+ minimizer and when $\lambda = 1$, we are at MSGD minimizer. It shows clearly that along the direction from SINTR+ minimizer to MSGD minimizer, SINTR+ achieve a flatter minimizer (see the three curves around $\lambda = 0$). The other two experiments is design to test gradient norm, loss objective value and training accuracy by letting $\theta(\alpha) = \theta + \alpha\xi$, where ξ is a normalized vector sampled from the unit ball with normal distribution. In both figures, 20 random directions are adding to both SINTR+ minimizer and MSGD minimizer. One could see clearly that SINTR+ minimizer is flatter and more robust than MSGD minimizer.

Thinking about the Taylor expansion of the loss function near the saddle point, for (stochastic) gradient based algorithm, we could end up with a point where the norm of gradient is close to zero (near the first-order critical point). As we do not have information of second order term, therefore, by the expansion, with a small perturbation of the saddle

point, we still may have the dramatic change of loss function value and gradient. However, this is not the case if we could minimize the loss function till second order critical point, where the Hessian is small, and therefore the second order term is nearly zero. This could be also verified by the next section.

3.5.3 Eigenvalue Evolution Along the Training Process

Considering that computing the exact full Hessian is computationally demanding, therefore we train on dataset sub-mnist and sub-cifra10, and limit the size of the network structure used in the experiment. Starting from a nearly saddle point where the norm of gradient is quite small, our algorithm would be able to escape in first 100 epochs in both case. If we check the right figure of top row, where the norm of positive eigenvalues $\|\Lambda^+\|$ and norm of negative eigenvalues $\|\Lambda^-\|$ of full Hessian at each iterative points. Obviously, as it is shown in Figure 3.5, both value reduce to around zero, which means the Hessian goes to zero along the training process.

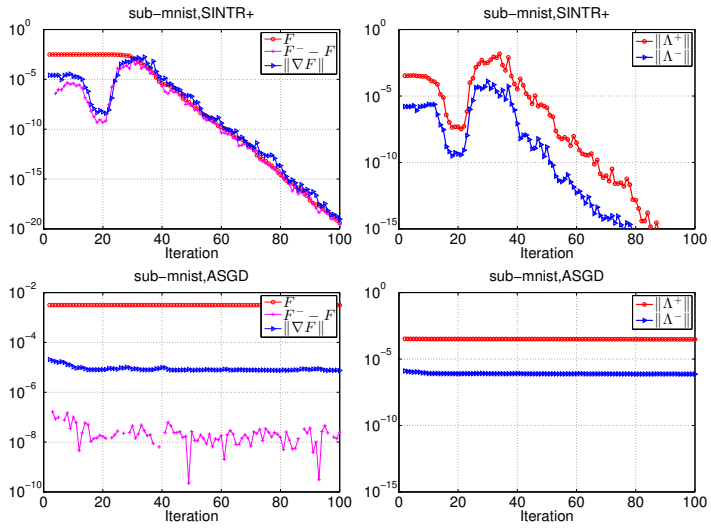


Figure 3.5: The evolution of eigenvalues for SINTR+ and ASGD on sub-mnist dataset

3.5.4 Accelerated SINTR with Adding Momentum

Note that in SINTR+ algorithm, we also consider to add momentum. This is motivated by the following observation when we look at the angle evolution between two adjacent

iterative points. In Figure 3.1, we apply Algorithm 3.2 and Algorithm 3.4 on sub-cifra10, respectively. At the first two rows, we record objective value, norm of gradient, and training accuracy for both algorithms, where it is clear to see the escaping happens. Moreover, it shows consistently that before the escaping happen, the angle between two adjacent point are quite small and when escaping happens, there is a dramatic change of the angles. This observation motivates us to have a large momentum to speed-up the escaping efficiency. Therefore, we add the momentum and make it adjustable. At each iteration, we would always use the maximum allowed momentum measured by objective value reduction.

3.5.5 Performance Comparison on the Full Dataset

In this Section, we compare various algorithms on full dataset CIFRA10 as shown in Figure 3.6. The detailed network structure is showing in Table 3.1 and Appendix. In figure 3.6, CIFRA10 is tested on different methods with their best hyper-parameters tuning. The stochastic Hessian-vector is evaluated based on 64 samples at each iterations.

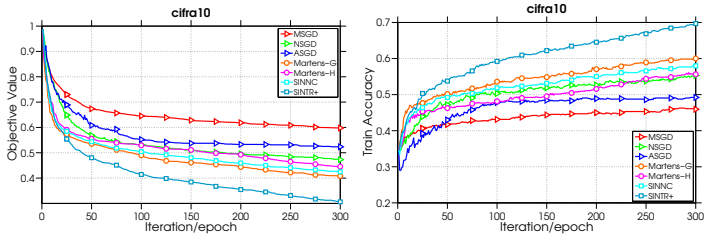


Figure 3.6: Objective value and Training error evolution on various methods.

3.5.6 The eigenvalue distribution evolution for SINTR+ on different dataset

We show the eigenvalue distribution evolution of full Hessian along the training process in the following Figures 3.7 and 3.8. Considering that to evaluate the exact eigenvalues is very computation demanding, we therefore do experiments on down-sampled dataset, i.e. sub-MNIST and sub-CIFRA10. We train two different models up to 100 iterations. The objective loss value and norm of positive/negative eigenvalues are reported at the first row of each figure. The second and third row collected the evolution of full Hessian eigenvalues at each iteration.

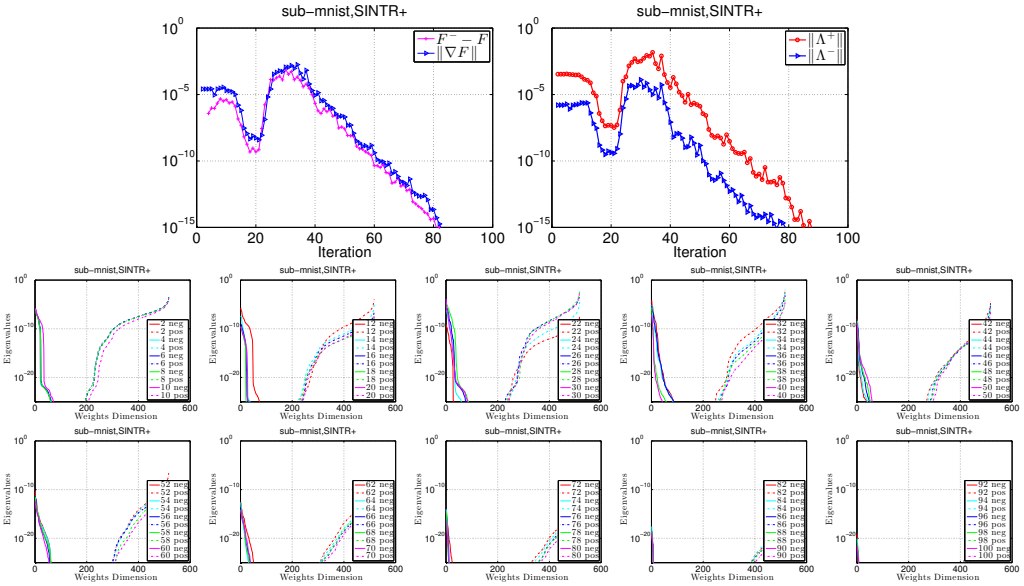


Figure 3.7: The distribution of eigenvalues for SINTR+ on sub-MNIST dataset, where extra line search and momentum is used. The legend, for example *42 neg*, means the negative eigenvalues distribution at 42 iteration.

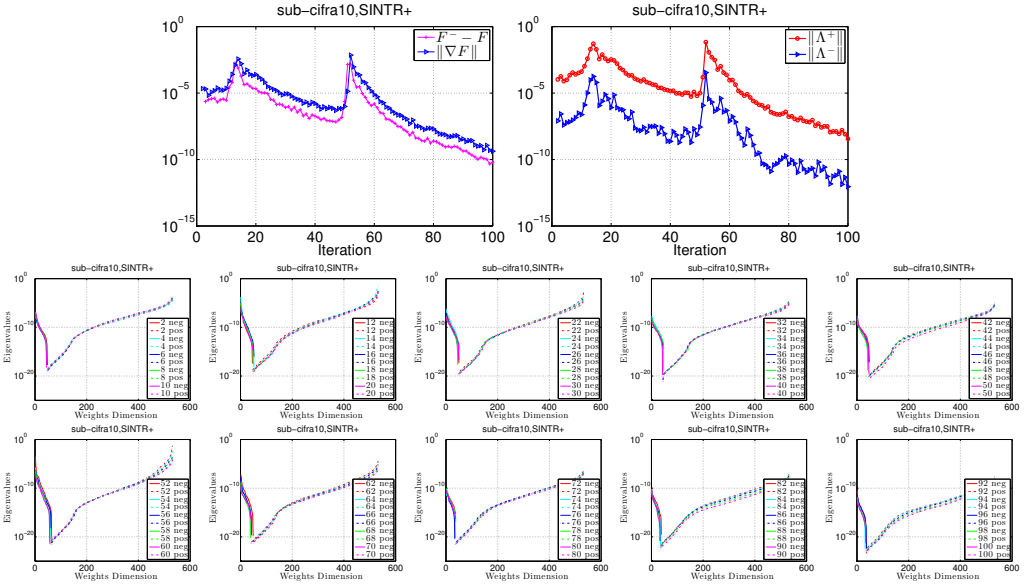


Figure 3.8: The distribution of eigenvalues for SINTR+ on sub-CIFRA10 dataset, where extra line search and momentum is used. The legend, for example *42 neg*, means the negative eigenvalues distribution at 42 iteration.

3.5.7 Discussion of Results

All of our results indicate that learning a deep neural networks in second order methods can be achieved effectively and performed better than first-order methods. Since the traditional line search technique does not apply for stochastic optimization algorithms, the common

practice in MSGD is either to use a diminishing step size, or to tune a fixed step size by hand, which can be time consuming in practice. Within second order methods, we could avoid the delicate tuning of hyper-parameters and pre-training.

3.6 Conclusion

In this chapter, we build steps to use second order method for training deep neural network efficiently. A new second order trust region optimization method is proposed to help escaping saddle point and achieve better accuracy. Note that our method does not require the approximation of Hessian to be semi-positive definite and fewer heuristics is needed to achieve a good performance. It will reduce the effect of tuning hyper-parameters. Further competitive performance of second order methods would need to utilize the distributed/-parallel setting [99]. We also have the preliminary result on the naive distributed setting at [28], which shows a potentially good performance.

Chapter 4

Efficient Distributed Hessian Free Algorithm for Large-scale Empirical Risk Minimization via Accumulating Sample Strategy

4.1 Introduction

In the field of machine learning, solving the expected risk minimization problem has received lots of attentions over the last decades, which is in the form of

$$\min_{w\in\mathbb{R}^d} L(w) = \min_{w\in\mathbb{R}^d} \mathbb{E}_z[f(w,z)], \tag{4.1}$$

where z is a $d+1$ dimensional random variable containing both feature variables and a response variable. $f(w,z)$ is a loss function with respect to w and any fixed value of z.

In most practical problems, the distribution of z is either unknown or leading great difficulties to evaluate the expected loss. One general idea is to estimate the expectation with a statistical average over a large number of independent and identically distributed data samples of z, which is denoted by $\{z_1, z_2, \ldots, z_N\}$ where N is the total number of

samples. Thus, the problem in (4.1) can be rewritten as the Empirical Risk Minimization (ERM) problem

$$\min_{w\in\mathbb{R}^d} L_N(w) = \min_{w\in\mathbb{R}^d} \frac{1}{N}\sum_{i=1}^{N} f_i(w), \tag{4.2}$$

where $f_i(w) = f(w, z_i)$.

A lot of studies have been done on developing optimization algorithms to find an optimal solution of above problem under different setting. For example, [4, 61, 21, 48] are some of the gradient-based methods which require at least one pass over all data samples to evaluate the gradient $\nabla L_N(w)$. As the sample size N becomes larger, these methods would be less efficient compared to stochastic gradient methods where the gradient is approximated based on a small number of samples [76, 19, 86, 42, 63].

Second order methods are well known to share faster convergence rate by utilizing the Hessian information. Recently, several papers [10, 80, 55] have studied how to apply second orders methods to solve ERM problem. However, evaluating the Hessian inverse or a good approximation of it is always computationally costly, leading to a significant difficulty on applying these methods on large scale problems.

The above difficulty can be addressed by applying the idea of adaptive sample size methods [56, 23, 54], which is based on the following two facts. First, the empirical risk and the statistical loss have different minimizers, and it is not necessary to go further than the difference between the mentioned two objectives, which is called *statistical accuracy* . More importantly, a smaller ERM problem with a smaller subset of samples should have a solution which is close to the solution of the problem with full samples.

Following the idea of adaptive sample size, the complexity of Newton's method can be reduced [54] if the dimension d is small, but it is impractical to compute the Hessian inverse for large dimensional problems. In order to decrease the cost of computing the Hessian inverse, [23] proposed the k-Truncated Adaptive Newton (k-TAN) approach. In this method, the inverse of such approximated Hessian is calculated by increasing the sample size adaptively and using a rank-k approximation of the Hessian. The cost per iteration is $\mathcal{O}((\log k + n)d^2)$. Again, note that either when d is large, or in the case when k is close to d, this method can be quite inefficient. In this method, the Hessian approximation is done by truncating the

eigenvalue decomposition of the Hessian which requires to store the full Hessian matrix for calculating the k-largest eigenvalues and associated eigenvectors. Therefore, calculation of the Newton step, which needs the computation of Hessian inverse in the mentioned methods, cannot be practical for high dimensional problems. The number of samples in new empirical risk is geometrically increased by rate of $\alpha > 1$. This problem is again solved until its statistical accuracy, and this process repeats till the number of samples is at least equal to the number of full samples, and the final solution is the one with the error not larger than the statistical accuracy of the full dataset. Therefore, utilizing these two features results in lower computational complexity.

In this chapter, we propose an increasing sample size second-order method which solves the Newton step in ERM problems more efficiently. Our proposed algorithm, called Distributed Accumulated Newton Conjugate gradiEnt (DANCE), starts with a small number of samples and minimizes their corresponding ERM problem. This subproblem is solved up to a specific accuracy, and the solution of this stage is used as a warm start for the next stage in which we solve the next empirical risk with a larger number of samples, which contains all the previous samples. Such procedure is run iteratively until either all the samples have been included, or we find that it is unnecessary to further increase the sample size. Our DANCE method combines the idea of increasing sample size and the inexact damped Newton method discussed in [103] and [50]. Instead of solving the Newton system directly, we apply preconditioned conjugate gradient (PCG) method as the solver for each Newton step. Also, it is always a challenging problem to run first order algorithms such as SGD and Adam [38] in a distributed fashion. The DANCE method is designed to be easily parallelized and shares the strong scaling property, i.e., linear speed-up property. Since it is possible to split gradient and Hessian-vector product computations across different machines, it is always expected to get extra acceleration via increasing the number of computational nodes. We formally characterize the required number of communication rounds in order to reach the statistical accuracy of the full dataset.We show that, under distributed setting, DANCE is communication efficient in both theory and experiments.

We organize this chapter as following. In Section 4.2, we introduce the necessary assumptions and the definition of statistical accuracy. Section 3 describes the proposed algorithm

and its distributed version. Section 4 explores the theoretical guarantees on complexity of DANCE. In Section 5, we demonstrate the outstanding performance of our algorithm in practice. In Section 6, we close the chapter by concluding remarks.

4.2 Problem Formulation

In this chapter, we focus on finding the optimal solution w^* of the problem in (4.1). As described earlier, due to difficulties in the expected risk minimization, as an alternative, we aim to find a solution for the empirical loss function $L_N(w)$, which is the empirical mean over N samples. Now, consider the empirical loss $L_n(w)$ associated with $n \leq N$ samples. In [8] and [6], it has been shown that the difference between the expected loss and the empirical loss L_n with high probability (w.h.p.) is upper bounded by the statistical accuracy V_n, i.e., w.h.p.

$$\sup_{w \in \mathbb{R}^d} |L(w) - L_n(w)| \leq V_n. \tag{4.3}$$

In other words, there exists a constant ϑ such that the inequality (4.3) holds with probability of at least $1 - \vartheta$. Generally speaking, statistical accuracy V_n depends on n (although it depends on ϑ too, but for simplicity in notation we just consider the size of the samples), and is of the order $V_n = \mathcal{O}(1/n^\gamma)$ where $\gamma \in [0.5, 1]$ [96, 7, 3].

For problem (4.2), if we find an approximate solution w_n which satisfies the inequality $L_n(w_n) - L_n(\hat{w}_n) \leq V_n$, where $\hat{w}_n$ is the true minimizer of L_n, it is not necessary to go further and find a better solution (a solution with less optimization error). The reason comes from the fact that for a more accurate solution the summation of estimation and optimization errors does not become smaller than V_n. Therefore, when we say that w_n is a V_n-suboptimal solution for the risk L_n, it means that $L_n(w_n) - L_n(\hat{w}_n) \leq V_n$. In other words, w_n solves problem (4.2) within its statistical accuracy.

It is crucial to note that if we add an additional term in the magnitude of V_n to the empirical loss L_n, the new solution is also in the similar magnitude as V_n to the expected loss L. Therefore, we can regularize the non-strongly convex loss function L_n by $cV_n\|w\|^2/2$

and consider it as the following problem:

$$\min_{w\in\mathbb{R}^d} R_n(w) := \frac{1}{n}\sum_{i=1}^{n} f_i(w) + \frac{cV_n}{2}\|w\|^2. \tag{4.4}$$

The noticeable feature of the new empirical risk R_n is that R_n is cV_n-strongly convex, where c is a positive constant. Thus, we can utilize any practitioner-favorite algorithm. Specifically, we are willing to apply the inexact damped Newton method, which will be discussed in the next section. Due to the fact that a larger strong-convexity parameter leads to a faster convergence, we could expect that the first few steps would converge fast since the values of cV_n in these steps are large (larger statistical accuracy), as will be discussed in Theorem 4.4.4. From now on, when we say w_n is an V_n-suboptimal solution of the risk R_n, it means that $R_n(w_n) - R_n(w_n^*) \leq V_n$, where w_n^* is the true optimal solution of the risk R_n. Our final aim is to find w_N which is V_N-optimal solution for the risk R_N which is the risk over the whole dataset.

4.3 Distributed Accumulated Newton-CG Method

The goal in inexact damped Newton method, as discussed in [103], is to find the next iterate based on an approximated Newton-type update. It has two important differences comparing to Newton's method. First, as it is clear from the word "damped", the learning rate of the inexact damped Newton type update is not 1, since it depends on the approximation of Newton decrement. The second distinction is that there is no need to compute exact Newton direction (which is very expensive to calculate in one step). Alternatively, an approximated inexact Newton direction is calculated by applying an iterative process to obtain a direction with desirable accuracy under some measurements.

In order to utilize the important features of ERM, we combine the idea of increasing sample size and the inexact damped Newton method. In our proposed method, we start with handling a small number of samples, assume m_0 samples. We then solve its corresponding ERM to its statistical accuracy, i.e. V_{m_0}, using the inexact damped Newton algorithm. In the next step, we increase the number of samples geometrically with rate of $\alpha > 1$, i.e., αm_0

samples. The approximated solution of the previous ERM can be used as a warm start point to find the solution of the new ERM. The sample size increases until it equals the number of full samples.

Consider the iterate w_m within the statistical accuracy of the set with m samples, i.e. $\mathcal{S}_m$ for the risk R_m. In DANCE, we increase the size of the training set to $n = \alpha m$ and use the inexact damped Newton to find the iterate w_n which is V_n-suboptimal solution for the sample set $\mathcal{S}_n$, i.e. $R_n(w_n) - R_n(w_n^*) \leq V_n$ after K_n iterations. To do so, we initialize $\tilde{w}_0 = w_m$ and update the iterates according to the following

$$\tilde{w}_{k+1} = \tilde{w}_k - \tfrac{1}{1+\delta_n(\tilde{w}_k)} v_k, \tag{4.5}$$

where v_k is an ϵ_k-Newton direction. The outcome of applying the update in (4.5) for $k = K_n$ iterations is the approximate solution w_n for the objective function R_n, i.e., $w_n := \tilde{w}_{K_n}$.

To properly define the approximate Newton direction v_k, first consider that the gradient and Hessian of the objective function R_n can be evaluated as

$$\nabla R_n(w) = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(w) + cV_n w \tag{4.6}$$

and

$$\nabla^2 R_n(w) = \frac{1}{n} \sum_{i=1}^{n} \nabla^2 f_i(w) + cV_n I, \tag{4.7}$$

respectively.

Indeed, the favorable descent direction would be the Newton direction

$$-\nabla^2 R_n(\tilde{w}_k)^{-1} \nabla R_n(\tilde{w}_k);$$

however, the cost of computing this direction is prohibitive. Therefore, we use v_k which is an ϵ_k-Newton direction satisfying the condition

$$\|\nabla^2 R_n(\tilde{w}_k) v_k - \nabla R_n(\tilde{w}_k)\| \leq \epsilon_k. \tag{4.8}$$

As we use the descent direction v_k which is an approximation for the Newton step, we also

redefine the Newton decrement $\delta_n(\tilde{w}_k)$ based on this modification. To be more specific, we define

$$\delta_n(\tilde{w}_k) := (v_k^T \nabla^2 R_n(\tilde{w}_k) v_k)^{1/2}$$

as the approximation of (exact) Newton decrement $(\nabla R_n(\tilde{w}_k)^T \nabla^2 R_n(\tilde{w}_k)^{-1} \nabla R_n(\tilde{w}_k))^{1/2}$, and use it in the update in (4.5).

In order to find v_k which is an ϵ_k-Newton direction, we use Preconditioned CG (PCG). As it is discussed in [103, 66], PCG is an efficient iterative process to solve Newton system with the required accuracy. The preconditioned matrix that we considered is in the form of $P = \tilde{H}_n + \mu_n I$, where $\tilde{H}_n = \frac{1}{|\mathcal{A}_n|} \sum_{i \in \mathcal{A}_n} \nabla^2 R_n^i(w)$, $\mathcal{A}_n \subset \mathcal{S}_n$, and μ_n is a small regularization parameter. In this case, v_k is an approximate solution of the system $P^{-1}\nabla^2 R_n(\tilde{w}_k) v_k = P^{-1}\nabla R_n(\tilde{w}_k)$. The reason for using preconditioning is that the condition number of matrix $P^{-1}\nabla^2 R_n(\tilde{w}_k)$ may be close to 1 in the case when $\tilde{H}_n$ is close to $\nabla^2 R_n(\tilde{w}_k)$; consequently, PCG can be faster than CG. The PCG steps are summarized in Algorithm 4.2. In every iteration of Algorithm 4.2, a system needs to be solved in step 10. Due to the structure of matrix P, and as it is discussed in [50], this matrix can be considered as $|\mathcal{A}_n|$ rank 1 updates on a diagonal matrix, and now, using Woodbury Formula [71] is a very efficient way to solve the mentioned system. The following lemma states the required number of iterations for PCG to find an ϵ_k-Newton direction v_k.

Lemma 4.3.1. *(Lemma 4 in [103]) Suppose Assumption 4.4.2 holds and* $\|\tilde{H}_n - \nabla^2 R_n(\tilde{w}_k)\| \leq \mu_n$. *Then, Algorithm 4.2, after* $C_n(\epsilon_k)$ *iterations calculates* v_k *such that* $\|\nabla^2 R_n(\tilde{w}_k) v_k - \nabla R_n(\tilde{w}_k)\| \leq \epsilon_k$, *where*

$$C_n(\epsilon_k) = \left\lceil \sqrt{(1 + \tfrac{2\mu_n}{cV_n})} \log \left(\frac{2\sqrt{\frac{cV_n + L}{cV_n}} \|\nabla R_n(\tilde{w}_k)\|}{\epsilon_k} \right) \right\rceil . \tag{4.9}$$

Note that ϵ_k has a crucial effect on the speed of the algorithm. When $\epsilon_k = 0$, then v_k is the exact Newton direction, and the update in (4.5) is the exact damped Newton step (which recovers the update in Ada Newton algorithm in [54] when the step-length is

Algorithm 4.1 DANCE

1: Initialization: Sample size increase constant α, initial sample size $n = m_0$ and $w_n = w_{m_0}$ with $\|\nabla R_n(w_n)\| < (\sqrt{2c})V_n$
2: **while** $n \leq N$ **do**
3: Update $w_m = w_n$ and $m = n$
4: Increase sample size: $n = \min\{\alpha m, N\}$
5: Set $\tilde{w}_0 = w_m$ and set $k = 0$
6: **repeat**
7: Calculate v_k and $\delta_n(\tilde{w}_k)$ by **Algorithm 4.2 PCG**
8: Set $\tilde{w}_{k+1} = \tilde{w}_k - \frac{1}{1+\delta_n(\tilde{w}_k)} v_k$
9: $k = k + 1$
10: **until** satisfy stop criteria leading to $R_n(\tilde{w}_k) - R_n(w_n^*) \leq V_n$
11: Set $w_n = \tilde{w}_k$
12: **end while**

Algorithm 4.2 PCG

1: **Input:** $\tilde{w}_k \in \mathbb{R}^d$, ϵ_k, and $\mathcal{A}_n$
2: Let $H = \nabla^2 R_n(\tilde{w}_k)$, $P = \frac{1}{|\mathcal{A}_n|} \sum_{i \in \mathcal{A}_n} \nabla^2 R_n^i(\tilde{w}_k) + \mu_n I$
3: Set $r^{(0)} = \nabla R_n(\tilde{w}_k)$, $u^{(0)} = s^{(0)} = P^{-1} r^{(0)}$
4: Set $v^{(0)} = 0$, $t = 0$
5: **repeat**
6: Calculate $Hu^{(t)}$ and $Hv^{(t)}$
7: Compute $\gamma_t = \frac{\langle r^{(t)}, s^{(t)} \rangle}{\langle u^{(t)}, Hu^{(t)} \rangle}$
8: Set $v^{(t+1)} = v^{(t)} + \gamma_t u^{(t)}$, $r^{(t+1)} = r^{(t)} - \gamma_t H u^{(t)}$
9: Compute $\beta_t = \frac{\langle r^{(t+1)}, s^{(t+1)} \rangle}{\langle r^{(t)}, s^{(t)} \rangle}$
10: Set $Ps^{(t+1)} = r^{(t+1)}$, $u^{(t+1)} = s^{(t+1)} + \beta_t u^{(t)}$
11: Set $t = t + 1$
12: **until** $\|r^{t+1}\| \leq \epsilon_k$
13: **Output:** $v_k = v^{(t+1)}$, $\delta_n(\tilde{w}_k) = \sqrt{v_k^T H v^{(t)} + \gamma_t v_k^T H u^{(t)}}$

1). Furthermore, the number of total iterations to reach V_N-optimal solution for the risk R_N is $\mathbf{K}$, i.e. $\mathbf{K} = K_{m_0} + K_{\alpha m_0} + \cdots + K_N$. It means that when we start with the iterate w_{m_0} with corresponding m_0 samples, after $\mathbf{K}$ iterations, we reach the point w_N with statistical accuracy of V_N for the whole dataset. In Theorem 4.4.4, the required rounds of communication to reach the mentioned statistical accuracy will be discussed.

Our proposed method is summarized in Algorithm 4.1. We start with m_0 samples, and an initial point w_{m_0} which is an $V_{m_0}-$ suboptimal solution for the risk R_{m_0}. In every iteration of outer loop of Algorithm 4.1, we increase the sample size geometrically with rate of α in step 4. In the inner loop of Algorithm 4.1, i.e. steps 6-10, in order to calculate the approximate Newton direction and approximate Newton decrement, we use PCG algorithm

which is shown in Algorithm 4.2. This process repeats till we get the point w_N with statistical accuracy of V_N.

Stopping Criteria Here we discuss two stopping criteria to fulfill the 10^{th} line of Algorithm 4.1. At first, considering w_n^* is unknown in practice, we can use strong convexity inequality as $R_n(\tilde{w}_k) - R_n(w_n^*) \leq \frac{1}{2cV_n}\|\nabla R_n(\tilde{w}_k)\|^2$ to find a stopping criterion for the inner loop, which satisfies $\|\nabla R_n(\tilde{w}_k)\| < (\sqrt{2c})V_n$. Another stopping criterion is discussed by [103], using the fact that the risk R_n is self-concordant. This criterion[1] can be written as $\delta_n(\tilde{w}_k) \leq (1-\beta)\sqrt{V_n}$, where $\beta \leq \frac{1}{20}$. The later stopping criterion implies that $R_n(\tilde{w}_k) - R_n(w_n^*) \leq V_n$ whenever $V_n \leq 0.68^2$.

Distributed Implementation Similar to the algorithm in [103], Algorithms 4.1 and 4.2 can also be implemented in a distributed environment. Suppose the entire dataset is stored across M machines, i.e., each machine stores N_i data samples such that $\sum_{i=1}^{M} N_i = N$. Under this setting, each iteration in Algorithm 4.1 can be executed on different machines in parallel with $\sum_{i=1}^{M} n_i = n$, where n_i is the batchsize on i^{th} machine. To implement Algorithm 4.2 in a distributed manner, a broadcast operation is needed at each iteration to guarantee that each machine will share the same $\tilde{w}_k$ value. Moreover, the gradient and Hessian-vector product can be computed locally and later reduce to the master machine. With the increasing of batch size, computation work on each machine will increase while we still have the same amount of communication need. As a consequence, the computation expense will gradually dominate the communication expense before the algorithm terminates. Therefore the proposed algorithm could take advantage of utilizing more machines to shorten the running time of Algorithm 4.2.

4.4 Complexity Analysis

In this section, first we define the self-concordant function, and after that we study the convergence properties of our algorithm. Self-concordant functions have the property that

[1] See section C.1.1 for the proof.

its third derivative can be controlled by its second derivative. By assuming that function $f : \mathbb{R}^d \to \mathbb{R}$ has continuous third derivative, we define self-concordant function as follows.

Definition 4.4.1. *A convex function $f : \mathbb{R}^d \to \mathbb{R}$ is M_f-self-concordant if for any $w \in dom(f)$ and $u \in \mathbb{R}^d$ we have*

$$|u^T(f'''(w)[u])u| \leq M_f(u^T\nabla^2 f(w)u)^{\frac{3}{2}}, \tag{4.10}$$

where $f'''(w)[u] := \lim_{t\to 0} \frac{1}{t}(\nabla^2 f(w+tu) - \nabla^2 f(w))$. As it is discussed in [61], any self-concordant function f with parameter M_f can be rescaled to become standard self-concordant (with parameter 2). Some of the well-known empirical loss functions which are self-concordant are linear regression, Logistic regression and squared hinge loss. In order to prove our results the following conditions are considered in our analysis.

Assumption 4.4.2. *The loss functions $f(w, z)$ are convex w.r.t w for all values of z. In addition, their gradients $\nabla f(w, z)$ are $L-$Lipschitz continuous*

$$\|\nabla f(w, z) - \nabla f(w', z)\| \leq L\|w - w'\|, \ \forall z. \tag{4.11}$$

Assumption 4.4.3. *The loss functions $f(w, z)$ are self-concordant w.r.t w for all values of z.*

The immediate conclusion of Assumption 4.4.2 is that both $L(w)$ and $L_n(w)$ are convex and L-smooth. Also, we can note that $R_n(w)$ is cV_n-strongly convex and $(cV_n + L)$-smooth. Moreover, by Assumption 4.4.3, $R_n(w)$ is also self-concordant. As it is discussed in [103] we use the following auxiliary function, which will be used in the analysis of the self-concordant function:

$$\omega(t) = t - \log(1 + t), \qquad t \geq 0, \tag{4.12}$$

In the rest of this section, we analyze the upper bound for the number of communication rounds needed to solve every subproblem up to its statistical accuracy.

We analyze the case when we have w_m which is a V_m-suboptimal solution of the risk

R_m, and we are interested in deriving a bound for the number of required communication rounds to ensure that w_n is a V_n-suboptimal solution for the risk R_n. We use the analysis of DiSCO algorithm discussed in [103] to find the mentioned bounds.

Theorem 4.4.4. *Suppose that Assumptions 4.4.2 and 4.4.3 hold. Consider w_m which satisfies $R_m(w_m) - R_m(w_m^*) \leq V_m$ and also the risk R_n corresponding to sample set $\mathcal{S}_n \supset \mathcal{S}_m$ where $n = \alpha m,\ \alpha > 1$. Set the parameter ϵ_k (the error in (4.8)) as following*[2]

$$\epsilon_k = \beta(\tfrac{cV_n}{L+cV_n})^{1/2}\|\nabla R_n(\tilde{w}_k)\|, \tag{4.13}$$

where $\beta \leq \frac{1}{20}$. Then, in order to find the variable w_n which is an V_n-suboptimal solution for the risk R_n, i.e $R_n(w_n) - R_n(w_n^*) \leq V_n$, the number of communication rounds T_n satisfies in the following:

$$T_n \leq K_n\left(1 + C_n(\epsilon_k)\right), \quad w.h.p. \tag{4.14}$$

where $K_n = \left\lceil \frac{R_n(w_m) - R_n(w_n^*)}{\frac{1}{2}\omega(1/6)} \right\rceil + \left\lceil \log_2(\frac{2\omega(1/6)}{V_n}) \right\rceil$. Here $\lceil t \rceil$ shows the smallest nonnegative integer larger than or equal to t.

As a result, the update in (4.5) needs to be done for $K_n = \mathcal{O}(\log_2 n)$ times in order to attain the solution w_n which is V_n-suboptimal solution for the risk R_n. Also, based on the result in (4.14), by considering the risk R_n, we can note that when the strong-convexity parameter for the mentioned risk (cV_n) is large, less number of iterations (communication rounds) are needed (or equally faster convergence is achieved) to reach the iterate with V_n-suboptimal solution; and this happens in the first steps.

Corollary 4.4.5. *Suppose that Assumptions 4.4.2 and 4.4.3 hold. Further, assume that w_m is a V_m-suboptimal solution for the risk R_m and consider R_n as the risk corresponding to sample set $\mathcal{S}_n \supset \mathcal{S}_m$ where $n = 2m$. If we set parameter ϵ_k (the error in (4.8)) as (4.13),*

[2]It is shown in [103] that with this tolerance, the inexact damped Newton method has linear convergence rate

then with high probability $\tilde{T}_n$ *communication rounds*

$$\tilde{T}_n \leq \Big(\Big\lceil \frac{\Big(3+\big(1-\frac{1}{2^\gamma}\big)\big(2+\frac{c}{2}\|w^*\|^2\big)\Big)V_m}{\frac{1}{2}\omega(1/6)} \Big\rceil + \Big\lceil \log_2\big(\frac{2\omega(1/6)}{V_n}\big) \Big\rceil \Big) \Big(1 + \Big\lceil \sqrt{1+\frac{2\mu}{cV_n}}\log_2\Big(\frac{2(cV_n+L)}{\beta c V_n}\Big) \Big\rceil \Big), \tag{4.15}$$

are needed to reach the point w_n *with statistical accuracy of* V_n *for the risk* R_n*.*

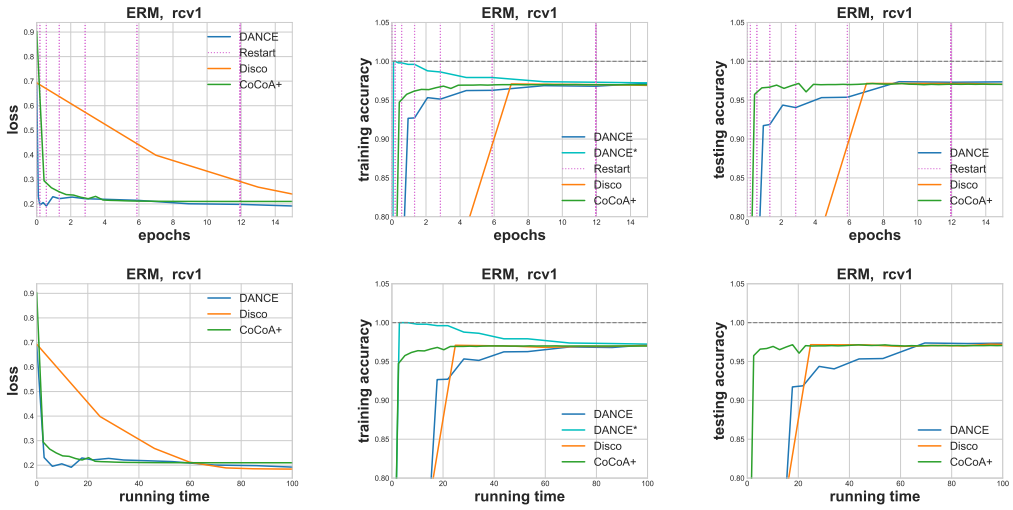


Figure 4.1: Performance of different algorithms on a Logistic Regression problem with rcv1 as dataset. For DANCE algorithm, we set $c = 0.1$ in (4.4), and the regularization parameter is set to be 10^{-4} for others. For figures in the middle where the y-axis represents training accuracy, the plot *DANCE* is the training accuracy based on the entire training set, while the plot *DANCE** represents the training accuracy based on the current sample size.

By Corollary 4.4.5, it is shown that[3] after $\tilde{\mathcal{T}}$ rounds of communication we reach a point with the statistical accuracy of V_N of the full training set, where $\tilde{\mathcal{T}}$ is bounded as following:

$$\tilde{\mathcal{T}} \leq \Bigg(2\log_2\frac{N}{m_0} + \Bigg(\frac{\Big(3+\big(1-\frac{1}{2^\gamma}\big)\big(2+\frac{c}{2}\|w^*\|^2\big)\Big)}{\frac{1}{2}\omega(1/6)} \frac{1-(\frac{1}{2^\gamma})^{\log_2\frac{N}{m_0}}}{1-\frac{1}{2^\gamma}} V_{m_0} \Bigg) + \log_2\frac{N}{m_0}\log_2\big(\frac{2\omega(1/6)}{V_N}\big) \Bigg) \Big(1 + \Big\lceil \sqrt{(1+\frac{2\mu}{cV_N})}\log_2\Big(\frac{2}{\beta}+\frac{2L}{\beta c}\cdot\frac{1}{V_N}\Big) \Big\rceil \Big) \quad \text{w.h.p.}, \tag{4.16}$$

where m_0 is the size of the initial training set. Note that the result in (4.16) implies that the

[3]The proof of this part is in section C.3.

overall rounds of communication to obtain the statistical accuracy of the full training set is of $\tilde{\mathcal{T}} = \mathcal{O}(\gamma(\log_2 N)^2\sqrt{N^\gamma}\log_2 N^\gamma)$. Hence, when $\gamma = 1$, we have $\tilde{\mathcal{T}} = \mathcal{O}((\log_2 N)^3\sqrt{N})$, and for $\gamma = 0.5$, the result is $\tilde{\mathcal{T}} = \mathcal{O}((\log_2 N)^3 N^{\frac{1}{4}})$. The rounds of communication for DiSCO algorithm in [103][4] is $\tilde{\mathcal{T}}_{DiSCO} = \mathcal{O}((R_N(w_0) - R_N(w_N^*) + \gamma(\log_2 N))\sqrt{N^\gamma}\log_2 N^\gamma)$ where $\gamma \in [0.5, 1]$. Comparing these bounds shows that the complexity of DANCE is independent of the choice of initial variable w_0 and the suboptimality $R_N(w_0) - R_N(w_N^*)$, while the overall complexity of DiSCO depends on the initial suboptimality. In addition, implementation of each iteration of DiSCO requires processing all the samples in the dataset, while DANCE only operates on an increasing subset of samples at each phase. Therefore, the computation complexity of DANCE is also lower than DiSCO for achieving the statistical accuracy of the training set.

4.5 Numerical Experiments

In this section, we present numerical experiments on several large real-world datasets to show that our restarting DANCE algorithm can outperform other existed methods on solving both convex and non-convex problems. Also, we compare the results from utilizing different number of machines to demonstrate the strong scaling property for our algorithm. All the algorithms are implemented in Python with PyTorch [68] library and we use MPI for Python [15] for setting distributed environment[5]. For all plots in this section, a vertical pink dashed lines represents a restarting in our DANCE algorithm. Note that the ERM loss function changes whenever a restarting is encountered.

Convex problems First, we compare our DANCE algorithm with two other distributed optimization algorithms CoCoA+ [49] and DiSCO [103], on solving convex problems. We choose these two algorithms in consideration of attaining a fair comparison between distributed first-order (CoCoA+) method and distributed second-order (DiSCO) approach. The experiments in this section are performed on a cluster with *16 Xeon E5-2620 CPUs (2.40GHz).*

[4] In order to have fair comparison, we put $f = R_N$, $\epsilon = V_N$, and $\lambda = cV_N$ in their analysis, and also the constants are ignored for the communication complexity.

[5] All codes to reproduce these experimental results are available at anonymous link.

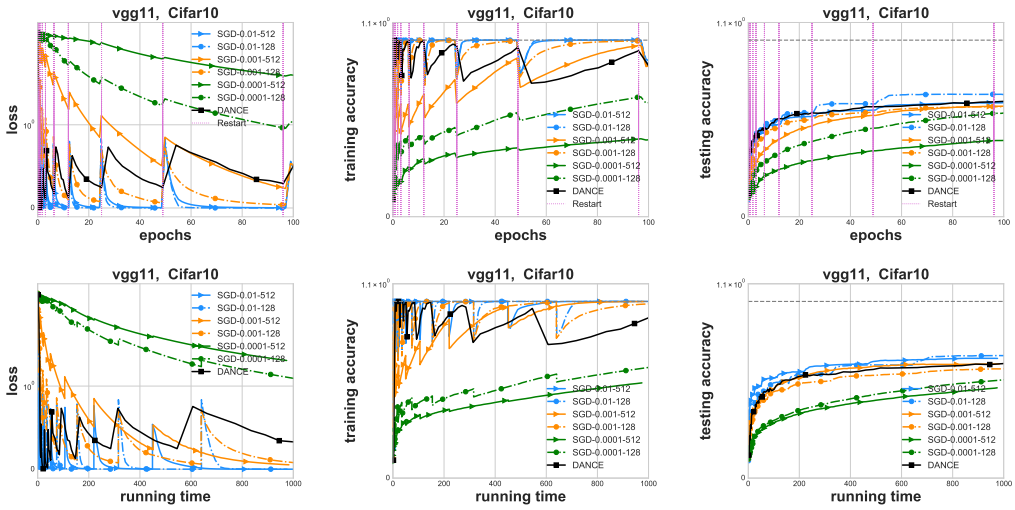


Figure 4.2: Comparison between DANCE and SGD with various hyper-parameters setting on Cifar10 dataset and vgg11 network. vgg11 represents [89] a 28 layers convolutional neural network (see details at Appendix C.4). Figures on the top and bottom show how loss values, training accuracy and test accuracy are changing with respect to epochs and running time. Note that we force both algorithms to restart (double training sample size) after achieving the following number of epochs: $0.2, 0.8, 1.6.3.2, 6.4, 12, 24, 48, 96$. For SGD, we varies learning rate from $0.01, 0.001, 0.0001$ and batchsize from $128, 512$. One could observe that SGD is sensitive to hyper-parameter settings, while DANCE has few hyper-parameters to tune but shows competitive performance.

In this chapter, we use logistic regression model on two binary classification tasks based on datasets *rcv1* and *gisette* [11] as our convex case. We leave the details of these two datasets in Appendix C.4. The two datasets is chosen following the principle from [103], since those two datasets show different relations between number of features and number of data samples (larger and smaller). We use logistic loss function defined as $f_i(w) := \log(1 + \exp(-y_i w^T x_i))$, where $x_i \in \mathbb{R}^d$ is data sample and $y_i \in \{-1, 1\}$ is binary label corresponding to $x_i, i \in [m]$. Then we minimize the empirical loss function as (4.4). Note that there is a fixed ℓ_2-regularization parameter $\lambda = 10^{-4}$ in DiSCO and CoCoA+ and we set $c = 0.1$ in (4.4) to form the ℓ_2-regularization parameter for our DANCE method.

We run our algorithm and compare algorithms with different datasets using 8 nodes. The starting batchsize on each node for our DANCE algorithm is set to 16 while other two algorithms go over the whole dataset at each iteration. For DANCE implementation, number of samples used to form the new ERM loss are doubled from previous iteration after each restarting. Furthermore, restarting happens whenever the norm of loss gradient

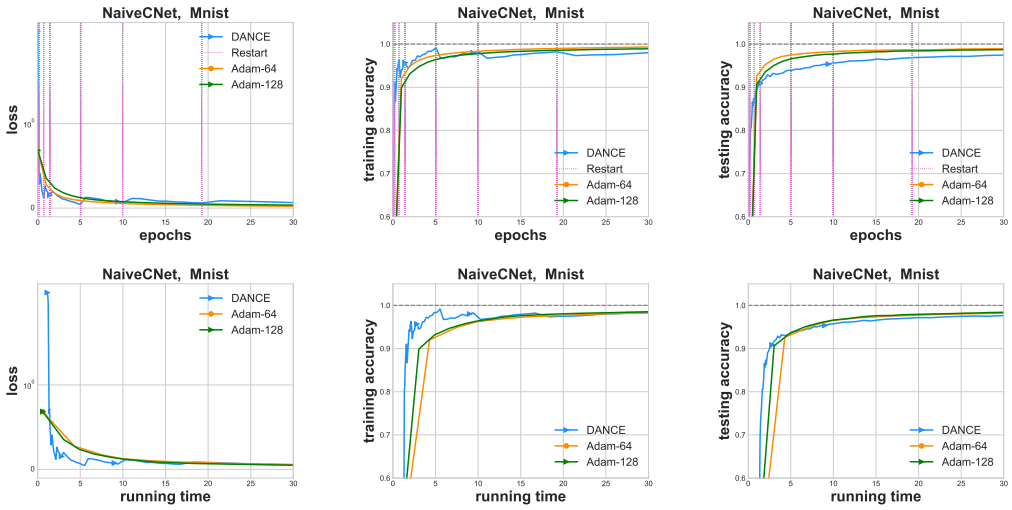


Figure 4.3: Comparison between DANCE and Adam on Mnist dataset and NaiveCNet. For DANCE, the initial batchsize is 1024. For Adam, the learning rate is 10^{-4} and the batchsize is either 64 or 128.
is lower than $1/\sqrt{m}$.

From Figure 4.1, we observe consistently that the DANCE algorithm has a better performance over the other two in the beginning stages. Both loss value and training accuracy under our DANCE algorithm converges to optimality by passing a small number of samples. It suggests that the DANCE can find a good solution in a warm starting manner regarding each restarting step. Compared with DiSCO, our restarting approach helps to reduce computation expense at the beginning iterations, where the second order methods usually performs less efficiently than the first order methods. Also, our algorithm converges fast when it is close to optimal solution, while the first order method become weak since the gradient vanishes around the optimal solution.

Non-convex problems Even though the complexity analysis in Section 4.4 only covers the convex case, the DANCE algorithm is also able to handle nonconvex problems efficiently. In this section, we compare our method with several stochastic first order algorithms, stochastic gradient descent (SGD), SGD with momentum (SGDMom), and Adam [38], on training convolution neural networks (CNNs) on two image classification datasets *Mnist* and *Cifar10*, we leave the details of datasets and the CNNs architecture applied on each dataset in Appendix C.4. To perform a fair comparison with those first order variants,

we assume the data comes in an online streaming manner, e.g., only a few data samples can be accessed at the beginning, and new data samples will come at a fixed rate. Such setting happens a lot in industrial production, where business data is collected as a streaming. We feed new data samples to all algorithms only if the amount of new data samples equals to the number of existed accessible data samples. The experiments in this section are run on an *AWS p2.xlarge instance with an NVIDIA K80 GPU*.

In Figure 4.2, we compare DANCE algorithm with the build-in SGD optimizer in pyTorch on Cifar dataset to train a 28 layers CNN (Vgg11) architecture. Note that there are several hyper-parameters we need to tune for SGD to reach the best performance, such as batch size and learning rate, which is not necessary for our DANCE algorithm. Since we have the online streaming data setting, we don't need to determine a restarting criterion. The results show that SGD is sensitive to hyper-parameters tuning, i.e., different combination of hyper-parameters affect the performance of SGD a lot and tune them well to achieve the best performance could be painful. However, our DANCE algorithm does not have such weakness and its performance is comparable to SGD with the best parameters setting. We also show that the DANCE algorithm leads to a faster decreasing on the loss value, which is similar to our convex experiments. Again, this is due to fast convergence rate of the second order methods. One could also found the additional experiments regarding the comparison with SGD with momentum and Adam in terms of Mnist with NaiveCNet at Appendix C.5.

Regarding Figure 4.3, the performance of build-in Adam optimizer and our DANCE algorithm are compared regarding Mnist dataset and a 4 layer NaiveCNet (see the details in Appendix C.4). In this experiment, we do not assume that the data samples follow an online streaming manner for Adam, i.e., the Adam algorithm does not have a restarting setting and therefore it runs on whole dataset directly. Also, this experiment is performed only on CPUs. We set the learning-rate for Adam as 10^{-4} and varies the running batch-size from 64 and 128. The evolution of loss, training accuracy, testing accuracy with respect to epochs and running time regarding the whole dataset are reported in Figure 4.3 for different algorithms. One could observe that under the same epochs, Adam eventually achieves the better testing accuracy, while if we look at running time, our DANCE algorithm would be faster due to the distributed implementation. The strong scaling property of our algorithm

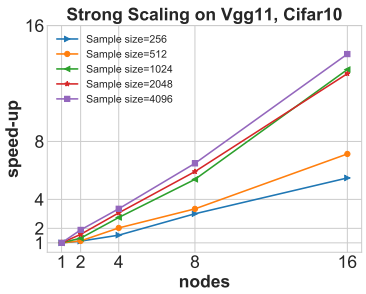


Figure 4.4: Performance of DANCE algorithm with different number of computing nodes. is also reported in the following experiment.

Strong scaling Finally, we demonstrate that our DANCE algorithm shares a strong scaling property. As shown in Figure 4.4, whenever we increase the number of nodes, we can always obtain acceleration towards optimality. We use the starting batchsize from 256 upto 4096, and the speed-up compared to serial run (1 node) is reported. It indicates that as we increase the batchsize, the speed-up becomes closer to ideal linear speed-up. Since our restarting approach will increase sampling size along the training process, after several restarting, we are able to reach a strong scaling performance asymptoticly.

4.6 Conclusion

We proposed an efficient distributed Hessian free algorithm DANCE with increasing sample size strategy to solve the empirical risk minimization problem. Our algorithm can converge to a low statistical accuracy in very few epochs and also be implemented in a distributed environment naturally. We analyzed the communication-efficiency of our algorithm, and showed that our algorithm is more efficient than DiSCO algorithm [103] in communication. Numerical experiments are presented to demonstrate the advantages of our proposed algorithm on both convex and non-convex problems.

Chapter 5

UCLibrary: A Unconstrained Optimization Library for Nonlinear Problems

5.1 Introduction

The UCLibrary[1] is a highly Python-based library of a collections of nonlinear optimization algorithms on unconstrained problems. It solves minimization problem of the form

$$\min_{x \in \mathbb{R}^n} f(x) \tag{5.1}$$

where $f \in \mathcal{C}^2$ and almost always nonconvex. It relies on the standard Cutest test problems set [25] and users are free and able to introduce new test functions easily.

5.2 Tour of the UCLibrary

Here we exemplify a instance as to solve a problem in Cutest problem set.

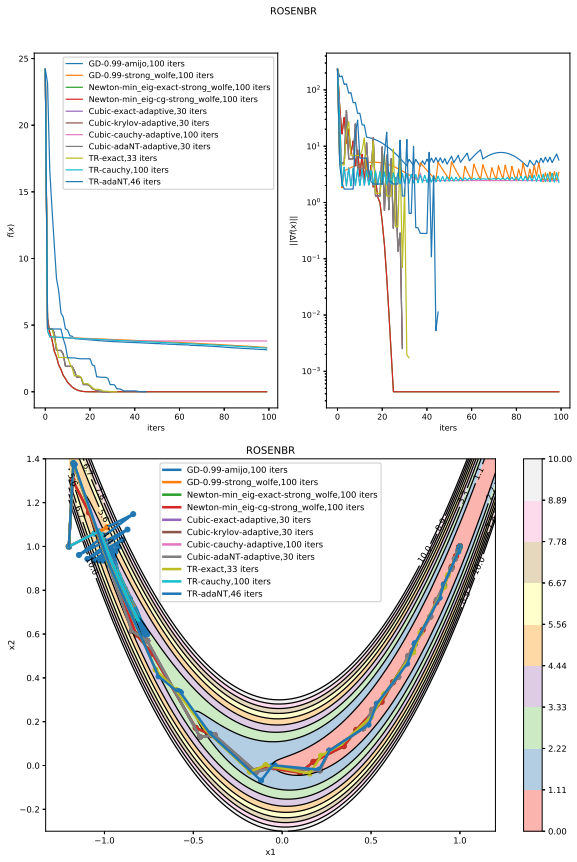
Single Run

[1] `https://bitbucket.org/xih314/mreleven/src/master/`

```
import ...
config.read('config.ini')
problem = CutestProblem('ROSENBR')
demo = Demo(problem)

problem.setInitialPoint()
optim = optimizers.Cubic(problem, lr=False,
                    mode='exact', adaptive=True)

for _ in range(config.max_iters):
    optim.step()
    if optim.terminationCondition(mode='first_order',
                                  tol=config.tol):
        break

demo.addContourTrace(optim)
demo.drawPerformancePlot(optim)
demo.showPlot()
__________________
Default initial point by Cutest...
Cubic-exact-adaptive
('cholesky_decomp_counter', 132)
('cholesky_linear_solver_counter', 292)
('hess_counter', 30)
('step_counter', 30)
('grad_counter', 30)
```

Multiple Run

```
import ...

def run(problem, optim, demo):
    for _ in range(config.max_iters):
        optim.step()
        if optim.terminationCondition(mode='first_order',
                                      tol=config.tol):
            break
    demo.addContourTrace(optim)
    demo.drawPerformancePlot(optim)

config.read('config.ini')
problem = CutestProblem('ROSENBR')
problem.setInitialPoint()

demo = Demo(problem)

for mode in ['exact', 'krylov']:
    optim = optimizers.Cubic(problem, lr=False,
                         mode=mode, adaptive=True)
    run(problem, optim, demo)

optim = optimizers.TrustRegion(problem, mode='cauchy')
run(problem, optim, demo)

demo.showPlot()
```



5.3 List of Main Modules

Gradient Based Solver, *optimizer.GD*

- Vanilla GD
- Nesterov's acceleration
- Heavy Ball acceleration
- Dynamic momentum
- Optimal momentum
- Static momentum
- Restart scheme

Practical Curvature based, *optimizer.Newton*

- Constant damping

- Levenberg–Marquardt damping
- Truncated hessian

Trust Region, *optimizer.TrustRegion*

- Vanilla TR

Cubic Regularization, *optimizer.Cubic*

- Vanilla CR
- Adaptive CR
- CRm (CR with momentum)

Line Search, *LineSearch*

- Backtracking (Amijo linesearch)
- Strong-wolfe

Subsolvers, *SubRoutine*

- Exact solver for positive definite matrix
- Cauchy point
- Dog-leg
- CG for positive positive definite matrix
- Steinghaug-Toint CG
- Generalized Lanczos trust region
- exact tridiagonal matrix subsolver
- exact regularized subproblem solver
- AdaNT

Chapter 6

Conclusion

Empirical risk minimization problems are important both in the theory and practical applications of machine learning. In this dissertation, we mainly studied several aspects for solving such problems.

The first is how adaptive sampling strategy can help improve the performance of ERMs. The answers could be revealed partially in Chapter 1 and Chapter 4. In Chapter 1, an importance sampling strategy is purposed for dual-free SDCA method, and then the strategy is generalized to mini-batch dual-free SDCA method. We show both in theoretical and empirical that using non-uniform adaptive sampling is beneficial in practice. In Chapter 4, an increasing sample size strategy is employed for optimizing the large-scaled ERMs. By considering that the decent performance of second-order method is usually achieved when the initial point is closer to the optimal, we purpose intuitively an increasing samples size strategy. We start from a relative small scale problem, which, is considering easy to solve. The output of the small scale problem could be then utilized as the starting point of next stage, with more samples including all samples from last stage. The efficiency of the purposed method is confirmed in the convex setting. Numerical experiments are done on both convex and nonconvex cases, which show a competitive performance comparing to the gradient based methods.

The second is how to handle the non-convexity raised from the nonconvex ERMs and/or training neural networks. In Chapter 1, we investigated dual-free SDCA on a special class of non-convex loss, where the single loss is non-convex but the average loss among all samples

is convex. The complexity results of convergence are derived under the setting. Meanwhile, since SDCA is gradient-based methods, therefore it is relative easy without annoying negative curvature. Move to a more general nonconvex family, i.e. in Chapter 2, Chapter 3, and Chapter 4, all the efforts are to make the matrix-free second-order methods works well in practical, especially when training neural networks. Chapter 2, Chapter 3 and Chapter 4 focus on variants of Newton-CG methods to address the negative curvature.

The last Chapter 5 introduces a python-based library for unconstrained optimization problem for all purposed methods in this dissertation (in deterministic setting) and also other relevant methods from literature. It builds a fair platform on comparing various methods for solving unconstrained optimization.

Bibliography

[1] Dario Amodei, Rishita Anubhai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Jingdong Chen, Mike Chrzanowski, Adam Coates, Greg Diamos, et al. Deep speech 2: End-to-end speech recognition in english and mandarin. *arXiv:1512.02595*, 2015.

[2] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv:1607.06450*, 2016.

[3] Peter L Bartlett, Michael I Jordan, and Jon D McAuliffe. Convexity, classification, and risk bounds. *Journal of the American Statistical Association*, 101(473):138–156, 2006.

[4] Amir Beck and Marc Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM journal on imaging sciences*, 2(1):183–202, 2009.

[5] Albert S Berahas, Jorge Nocedal, and Martin Takáč. A multi-batch l-bfgs method for machine learning. *arXiv:1605.06049*, 2016.

[6] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[7] olivier Bousquet. *Concentration Inequalities and Empirical Processes Theory Applied to the Analysis of Learning Algorithms*. PhD thesis, Biologische Kybernetik, 2002.

[8] Olivier Bousquet and Léon Bottou. The tradeoffs of large scale learning. In *Advances in Neural Information Processing Systems 20*, pages 161–168, 2008.

[9] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[10] Richard H Byrd, Samantha L Hansen, Jorge Nocedal, and Yoram Singer. A stochastic quasi-newton method for large-scale optimization. *SIAM Journal on Optimization*, 26(2):1008–1031, 2016.

[11] Chih-Chung Chang and Chih-Jen Lin. Libsvm: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(3):27, 2011.

[12] Anna Choromanska, Mikael Henaff, Michael Mathieu, Gérard Ben Arous, and Yann LeCun. The loss surfaces of multilayer networks. In *AISTATS*, 2015.

[13] Dominik Csiba, Zheng Qu, and Peter Richtárik. Stochastic dual coordinate ascent with adaptive probabilities. *In 32nd International Conference on Machine Learning*, 2015.

[14] Dominik Csiba and Peter Richtárik. Primal method for erm with flexible mini-batching schemes and non-convex losses. *arXiv:1506.02227*, 2015.

[15] Lisandro D Dalcin, Rodrigo R Paz, Pablo A Kler, and Alejandro Cosimo. Parallel distributed computing using python. *Advances in Water Resources*, 34(9):1124–1139, 2011.

[16] Dipankar Das, Sasikanth Avancha, Dheevatsa Mudigere, Karthikeyan Vaidynathan, Srinivas Sridharan, Dhiraj Kalamkar, Bharat Kaul, and Pradeep Dubey. Distributed deep learning using synchronous stochastic gradient descent. *arXiv:1602.06709*, 2016.

[17] Yann N Dauphin, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *Advances in neural information processing systems*, pages 2933–2941, 2014.

[18] Aaron Defazio, Francis Bach, and Simon Lacoste-Julien. Saga: A fast incremental gradient method with support for non-strongly convex composite objectives. In *Advances in Neural Information Processing Systems*, pages 1646–1654, 2014.

[19] Aaron Defazio, Francis Bach, and Simon Lacoste-Julien. Saga: A fast incremental gradient method with support for non-strongly convex composite objectives. In *Advances in Neural Information Processing Systems 27*, pages 1646–1654, 2014.

[20] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In *ICML*, pages 647–655, 2014.

[21] Dmitriy Drusvyatskiy, Maryam Fazel, and Scott Roy. An optimal first order method based on optimal quadratic averaging. *SIAM Journal on Optimization*, 28(1):251–271, 2018.

[22] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[23] Mark Eisen, Aryan Mokhtari, and Alejandro Ribeiro. Large scale empirical risk minimization via truncated adaptive Newton method. *arXiv preprint arXiv:1705.07957*, 2017.

[24] Rong Ge, Furong Huang, Chi Jin, and Yang Yuan. Escaping from saddle points—online stochastic gradient for tensor decomposition. In *Proceedings of The 28th Conference on Learning Theory*, pages 797–842, 2015.

[25] Nicholas IM Gould, Dominique Orban, and Philippe L Toint. Cutest: a constrained and unconstrained testing environment with safe threads for mathematical optimization. *Computational Optimization and Applications*, 60(3):545–557, 2015.

[26] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6645–6649. IEEE, 2013.

[27] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv:1512.03385*, 2015.

[28] Xi He, Dheevatsa Mudigere, Mikhail Smelyanskiy, and Martin Takáč. Distributed hessian-free optimization for deep neural network. *34th AAAI Workshop on Distributed Machine Learning*, 2017.

[29] Xi He and Martin Takáč. Dual free sdca for empirical risk minimization with adaptive probabilities. *NIPS Workshop on Optimization in Machine Learning*, 2015.

[30] Xi He, Rachael Tappenden, and Martin Takáč. Dual free adaptive minibatch sdca for empirical risk minimization. *Frontiers in Applied Mathematics and Statistics*, 4(33), 2018.

[31] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *Signal Processing Magazine, IEEE*, 29(6):82–97, 2012.

[32] Cho-Jui Hsieh, Kai-Wei Chang, Chih-Jen Lin, S Sathiya Keerthi, and Sellamanickam Sundararajan. A dual coordinate descent method for large-scale linear svm. In *Proceedings of the 25th international conference on Machine learning*, pages 408–415. ACM, 2008.

[33] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv:1502.03167*, 2015.

[34] Martin Jaggi, Virginia Smith, Martin Takáč, Jonathan Terhorst, Sanjay Krishnan, Thomas Hofmann, and Michael I Jordan. Communication-efficient distributed dual coordinate ascent. In *Advances in Neural Information Processing Systems*, pages 3068–3076, 2014.

[35] Majid Jahani, Xi He, Chenxin Ma, Aryan Mokhtari, Dheevatsa Mudigere, Alejandro Ribeiro, and Martin Takáč. Efficient distributed hessian free algorithm for large-scale empirical risk minimization via accumulating sample strategy. *arXiv preprint arXiv:1810.11507*, 2018.

[36] Rie Johnson and Tong Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in Neural Information Processing Systems*, pages 315–323, 2013.

[37] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv:1609.04836*, 2016.

[38] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[39] Ryan Kiros. Training neural networks with stochastic hessian-free optimization. *arXiv:1301.3641*, 2013.

[40] Jakub Konečný, Jie Liu, Peter Richtárik, and Martin Takáč. mS2GD: Mini-batch semi-stochastic gradient descent in the proximal setting. *arXiv preprint arXiv:1410.4744*, 2014.

[41] Jakub Konečný and Peter Richtárik. Semi-stochastic gradient descent methods. *arXiv:1312.1666*, 2013.

[42] Jakub Konečnỳ and Peter Richtárik. Semi-stochastic gradient descent methods. *Frontiers in Applied Mathematics and Statistics*, 3:9, 2017.

[43] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[44] Richard A Kronmal and Arthur V Peterson Jr. On the alias method for generating random variables from a discrete distribution. *The American Statistician*, 33(4):214–218, 1979.

[45] Yann LeCun and Yoshua Bengio. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[46] Jason D Lee, Max Simchowitz, Michael I Jordan, and Benjamin Recht. Gradient descent only converges to minimizers. In *Conference on Learning Theory*, pages 1246–1257, 2016.

[47] Ji Liu, Stephen J Wright, Christopher Ré, Victor Bittorf, and Srikrishna Sridhar. An asynchronous parallel stochastic coordinate descent algorithm. *The Journal of Machine Learning Research*, 16(1):285–322, 2015.

[48] Chenxin Ma, Naga Venkata C Gudapati, Majid Jahani, Rachael Tappenden, and Martin Takáč. Underestimate sequences via quadratic averaging. *arXiv preprint arXiv:1710.03695*, 2017.

[49] Chenxin Ma, Virginia Smith, Martin Jaggi, Michael I Jordan, Peter Richtárik, and Martin Takáč. Adding vs. averaging in distributed primal-dual optimization. *In 32th International Conference on Machine Learning, ICML 2015*, 2015.

[50] Chenxin Ma and Martin Takáč. Distributed inexact damped Newton method: Data partitioning and load-balancing. *arXiv preprint arXiv:1603.05191*, 2016.

[51] James Martens. Deep learning via hessian-free optimization. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 735–742, 2010.

[52] James Martens and Ilya Sutskever. Learning recurrent neural networks with hessian-free optimization. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 1033–1040, 2011.

[53] Dmytro Mishkin and Jiri Matas. All you need is a good init. *arXiv:1511.06422*, 2015.

[54] Aryan Mokhtari, Hadi Daneshmand, Aurelien Lucchi, Thomas Hofmann, and Alejandro Ribeiro. Adaptive Newton method for empirical risk minimization to statistical accuracy. In *Advances in Neural Information Processing Systems 29*, pages 4062–4070, 2016.

[55] Aryan Mokhtari and Alejandro Ribeiro. Global convergence of online limited memory bfgs. *Journal of Machine Learning Research*, 16(1):3151–3181, 2015.

[56] Aryan Mokhtari and Alejandro Ribeiro. First-order adaptive sample size methods to reduce complexity of empirical risk minimization. In *Advances in Neural Information Processing Systems 30*, pages 2057–2065, 2017.

[57] I Necoara and Dragos Clipici. Parallel random coordinate descent method for composite minimization. *submitted to SIAM Journal on Optimization*, 2013.

[58] Ion Necoara and Dragos Clipici. Efficient parallel coordinate descent algorithm for convex optimization problems with separable constraints: application to distributed mpc. *Journal of Process Control*, 23(3):243–253, 2013.

[59] Arvind Neelakantan, Luke Vilnis, Quoc V Le, Ilya Sutskever, Lukasz Kaiser, Karol Kurach, and James Martens. Adding gradient noise improves learning for very deep networks. *arXiv preprint arXiv:1511.06807*, 2015.

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

[61] Yurii Nesterov. *Introductory lectures on convex optimization: A basic course*, volume 87. Springer Science & Business Media, 2013.

[62] Yurii Nesterov and Boris T Polyak. Cubic regularization of newton method and its global performance. *Mathematical Programming*, 108(1):177–205, 2006.

[63] Lam M Nguyen, Jie Liu, Katya Scheinberg, and Martin Takáč. Sarah: A novel method for machine learning problems using stochastic recursive gradient. In *Proceedings of the 34th International Conference on Machine Learning*, pages 2613–2621, 2017.

[64] Atsushi Nitanda. Stochastic proximal gradient descent with acceleration techniques. In *Advances in Neural Information Processing Systems*, pages 1574–1582, 2014.

[65] Jorge Nocedal and Stephen Wright. *Numerical optimization*. Springer Science & Business Media, 2006.

[66] Jorge Nocedal and Stephen J Wright. *Sequential quadratic programming*. Springer, 2006.

[67] Alberto Olivares, Javier M Moguerza, and Francisco J Prieto. Nonconvex optimization using negative curvature within a modified linesearch. *European Journal of Operational Research*, 189(3):706–722, 2008.

[68] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[69] Barak A Pearlmutter. Fast exact multiplication by the hessian. *Neural computation*, 6(1):147–160, 1994.

[70] Boris T Polyak and Anatoli B Juditsky. Acceleration of stochastic approximation by averaging. *SIAM Journal on Control and Optimization*, 30(4):838–855, 1992.

[71] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical recipes: the art of scientific computing, 3rd Edition*. Cambridge University Press, 2007.

[72] Zheng Qu and Peter Richtárik. Coordinate descent with arbitrary sampling ii: Expected separable overapproximation. *arXiv preprint arXiv:1412.8063*, 2014.

[73] Zheng Qu, Peter Richtárik, and Tong Zhang. Quartz: Randomized dual coordinate ascent with arbitrary sampling. In *Advances in Neural Information Processing Systems*, pages 865–873, 2015.

[74] Peter Richtárik and Martin Takáč. Parallel coordinate descent methods for big data optimization. *Mathematical Programming*, pages 1–52, 2012.

[75] Peter Richtárik and Martin Takáč. Iteration complexity of randomized block-coordinate descent methods for minimizing a composite function. *Mathematical Programming*, 144(1-2):1–38, 2014.

[76] Nicolas L Roux, Mark Schmidt, and Francis R Bach. A stochastic gradient method with an exponential convergence _rate for finite training sets. In *Advances in Neural Information Processing Systems*, pages 2663–2671, 2012.

[77] Yousef Saad. *Iterative methods for sparse linear systems*. Siam, 2003.

[78] Tara N Sainath, Lior Horesh, Brian Kingsbury, Aleksandr Y Aravkin, and Bhuvana Ramabhadran. Accelerating hessian-free optimization for deep neural networks by implicit preconditioning and sampling. In *Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop on*, pages 303–308. IEEE, 2013.

[79] Mark Schmidt, Nicolas Le Roux, and Francis Bach. Minimizing finite sums with the stochastic average gradient. *arXiv preprint arXiv:1309.2388*, 2013.

[80] Nicol N Schraudolph, Jin Yu, and Simon Günter. A stochastic quasi-Newton method for online convex optimization. In *Artificial Intelligence and Statistics*, pages 436–443, 2007.

[81] Frank Seide, Hao Fu, Jasha Droppo, Gang Li, and Dong Yu. On parallelizability of stochastic gradient descent for speech dnns. In *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, pages 235–239. IEEE, 2014.

[82] S. Shalev-Shwartz and S. Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, New York, USA, 2014.

[83] Shai Shalev-Shwartz. SDCA without duality. *arXiv:1502.06177*, 2015.

[84] Shai Shalev-Shwartz, Yoram Singer, Nathan Srebro, and Andrew Cotter. Pegasos: Primal estimated sub-gradient solver for svm. *Mathematical programming*, 127(1):3–30, 2011.

[85] Shai Shalev-Shwartz and Tong Zhang. Stochastic dual coordinate ascent methods for regularized loss. *The Journal of Machine Learning Research*, 14(1):567–599, 2013.

[86] Shai Shalev-Shwartz and Tong Zhang. Stochastic dual coordinate ascent methods for regularized loss minimization. *Journal of Machine Learning Research*, 2013.

[87] Shai Shalev-Shwartz and Tong Zhang. Accelerated proximal stochastic dual coordinate ascent for regularized loss minimization. *Mathematical Programming*, pages 1–41, 2014.

[88] Christopher J Shallue, Jaehoon Lee, Joe Antognini, Jascha Sohl-Dickstein, Roy Frostig, and George E Dahl. Measuring the effects of data parallelism on neural network training. *arXiv preprint arXiv:1811.03600*, 2018.

[89] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv:1409.1556*, 2014.

[90] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[91] Trond Steihaug. The conjugate gradient method and trust regions in large scale optimization. *SIAM Journal on Numerical Analysis*, 20(3):626–637, 1983.

[92] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th international conference on machine learning (ICML-13)*, pages 1139–1147, 2013.

[93] Martin Takáč, Avleen Bijral, Peter Richtárik, and Nathan Srebro. Mini-batch primal and dual methods for svms. In *In 30th International Conference on Machine Learning, ICML 2013*, 2013.

[94] Martin Takáč, Peter Richtárik, and Nathan Srebro. Distributed mini-batch sdca. *arXiv preprint arXiv:1507.08322*, 2015.

[95] Rachael Tappenden, Martin Takáč, and Peter Richtárik. On the complexity of parallel coordinate descent. *arXiv preprint arXiv:1503.03033*, 2015.

[96] Vladimir Vapnik. *The nature of statistical learning theory*. Springer science & business media, 2013.

[97] Oriol Vinyals and Daniel Povey. Krylov subspace descent for deep learning. In *AISTATS*, pages 1261–1268, 2012.

[98] Simon Wiesler, Jinyu Li, and Jian Xue. Investigations on hessian-free optimization for cross-entropy training of deep neural networks. In *INTERSPEECH*, pages 3317–3321, 2013.

[99] Yue Yu, Jinrong Jiang, and Xuebin Chi. Using supercomputer to speed up neural network training. In *Parallel and Distributed Systems (ICPADS), 2016 IEEE 22nd International Conference on*, pages 942–947. IEEE, 2016.

[100] Yaxiang Yuan. On the truncated conjugate gradient method. *Mathematical Programming*, 87(3):561–573, 2000.

[101] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv:1212.5701*, 2012.

[102] Sixin Zhang. *Distributed stochastic optimization for deep learning.* PhD thesis, New York University, 2016.

[103] Yuchen Zhang and Xiao Lin. Disco: Distributed optimization for self-concordant empirical loss. In *Proceedings of the 32nd International Conference on Machine Learning*, pages 362–370, 2015.

[104] Yuchen Zhang and Lin Xiao. Disco: distributed optimization for self-concordant empirical loss. *Proceedings of the 32nd International Conference on International Conference on Machine Learning (ICML-15)*, pages 362–370, 2015.

[105] Peilin Zhao and Tong Zhang. Stochastic optimization with importance sampling for regularized loss minimization. *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 1–9, 2015.

Appendix A

Proofs in Chapter 1

A.1 Preliminaries and Technical Results

Recall that w^* denotes an optimum of (1.1) and define $\alpha_i^* = -\phi_i'(x_i^T w^*)$. To simplify the proofs we introduce the following variables

$$A^{(t)} = \tfrac{1}{n}\|\alpha^{(t)} - \alpha^*\|^2 \text{ and } B^{(t)} = \|w^{(t)} - w^*\|^2. \tag{A.1}$$

At the optimum w^*, it holds that $0 = \nabla P(w^*) = \frac{1}{n}\sum_{i=1}^n \phi_i'(x_i^T w^*)x_i + \lambda w^*$, so $w^* = \frac{1}{\lambda n}\sum_{i=1}^n \alpha_i^* x_i$. Define $u_i^{(t)} \overset{\text{def}}{=} -\phi_i'(x_i^T w^{(t)})$, and therefore we have $\kappa_i^{(t)} = \alpha_i^{(t)} - u_i^{(t)}$ and $u_i^* = \alpha_i^*$.

The following two lemmas will be useful when proving our main results.

Lemma A.1.1. *Let $A^{(t)}$ and $B^{(t)}$ be defined in* (A.1)*, and let $v_i = \|x_i\|^2$ for all $i \in [n]$. Then, conditioning on $\alpha^{(t)}$, the following hold for given θ:*

$$\mathbb{E}[A^{(t+1)}|\alpha^{(t)}] - A^{(t)} = -\theta A^{(t)} + \frac{\theta}{n}\sum_{i=1}^n \Big((u_i^{(t)} - \alpha_i^*)^2 - \big(1 - \tfrac{\theta}{p_i}\big)(\kappa_i^{(t)})^2\Big), \tag{A.2}$$

$$\mathbb{E}[B^{(t+1)}|\alpha^{(t)}] - B^{(t)} = -\tfrac{2\theta}{\lambda}(w^{(t)} - w^*)^T\nabla P(w^{(t)}) + \sum_{i=1}^n \frac{\theta^2 v_i}{n^2\lambda^2 p_i}\big(\kappa_i^{(t)}\big)^2. \tag{A.3}$$

Proof. Note that at iteration t, only coordinate i (of α) is updated, so

$$A^{(t+1)} = \tfrac{1}{n}\|\alpha^{(t+1)} - \alpha^*\|^2 \equiv \frac{1}{n}\sum_{j\neq i}(\alpha_j^{(t)} - \alpha_j^*)^2 + \frac{1}{n}(\alpha_i^{(t+1)} - \alpha_i^*)^2. \tag{A.4}$$

Using $((1-t)a + tb)^2 = (1-t)a^2 + tb^2 - t(1-t)(a-b)^2$, we have,

$$\begin{aligned}
A^{(t+1)} - A^{(t)} &\overset{(A.4)}{=} \tfrac{1}{n}(\alpha_i^{(t+1)} - \alpha_i^*)^2 - \tfrac{1}{n}(\alpha_i^{(t)} - \alpha_i^*)^2 \\
&= \tfrac{1}{n}\Big(\alpha_i^{(t)} - \tfrac{\theta}{p_i}\kappa_i^{(t)} - \alpha_i^*\Big)^2 - \tfrac{1}{n}(\alpha_i^{(t)} - \alpha_i^*)^2 \\
&= \tfrac{1}{n}\left(\big(1 - \tfrac{\theta}{p_i}\big)(\alpha_i^{(t)} - \alpha_i^*) + \tfrac{\theta}{p_i}(u_i^{(t)} - \alpha_i^*)\right)^2 - \tfrac{1}{n}(\alpha_i^{(t)} - \alpha_i^*)^2 \\
&= \tfrac{1}{n}\left(1 - \tfrac{\theta}{p_i}\right)(\alpha_i^{(t)} - \alpha_i^*)^2 + \tfrac{\theta}{np_i}(u_i^{(t)} - \alpha_i^*)^2 - \left(1 - \tfrac{\theta}{p_i}\right)\tfrac{\theta}{np_i}(\kappa_i^{(t)})^2 - \tfrac{1}{n}(\alpha_i^{(t)} - \alpha_i^*)^2 \\
&= -\tfrac{\theta}{np_i}(\alpha_i^{(t)} - \alpha_i^*)^2 + \tfrac{\theta}{np_i}\left((u_i^{(t)} - \alpha_i^*)^2 - \left(1 - \tfrac{\theta}{p_i}\right)(\kappa_i^{(t)})^2\right).
\end{aligned} \tag{A.5}$$

Taking expectation over $i \in [n]$, conditioned on $\alpha^{(t)}$, gives the first result.

To obtain the second result consider

$$\begin{aligned}
B^{(t+1)} - B^{(t)} &\overset{(A.1)}{=} \|w^{(t+1)} - w^*\|^2 - \|w^{(t)} - w^*\|^2 \\
&\overset{(1.9)}{=} \|w^{(t)} - \tfrac{\theta}{n\lambda p_i}\kappa_i^{(t)}x_i - w^*\|^2 - \|w^{(t)} - w^*\|^2 \\
&= -\tfrac{2\theta}{n\lambda p_i}\kappa_i^{(t)}x_i^T(w^{(t)} - w^*) + \tfrac{\theta^2 v_i}{n^2\lambda^2 p_i^2}(\kappa_i^{(t)})^2.
\end{aligned}$$

Recall that $\mathbb{E}[\frac{1}{np_i}\kappa_i^{(t)}x_i] = \nabla P(w^{(t)})$ by (1.11). Thus, taking expectation over $i \in [n]$, conditioned on $w^{(t)}$, gives (A.3). □

The following Lemma and proof are similar to [14, Lemma 4] and [85, Lemma 1].

Lemma A.1.2. *Assume that each ϕ_i is $\tilde{L}_i$-smooth and convex. Then, for every w*

$$\begin{aligned}
\frac{1}{\tilde{L}}\Big(\frac{1}{n}\sum_{i=1}^n \|\phi_i'(x_i^T w) - \phi_i'(x_i^T w^*)\|^2\Big) &\leq \frac{1}{n}\sum_{i=1}^n \frac{1}{\tilde{L}_i}\|\phi_i'(w^T x_i) - \phi_i'(x_i^T w^*)\|^2 \\
&\leq 2\left(P(w) - P(w^*) - \tfrac{\lambda}{2}\|w - w^*\|^2\right).
\end{aligned} \tag{A.6}$$

Proof. Let $z, z^* \in \mathbb{R}$. Define

$$g_i(z) \stackrel{\text{def}}{=} \phi_i(z) - \phi_i(z^*) - \phi_i'(z^*)(z - z^*). \tag{A.7}$$

Because ϕ_i is $\tilde{L}_i$-smooth, so too is g_i, which implies that for all $z, \hat{z} \in \mathbb{R}$,

$$g_i(z) \leq g_i(\hat{z}) + g_i'(\hat{z})(z - \hat{z}) + \tfrac{\tilde{L}_i}{2}(z - \hat{z})^2. \tag{A.8}$$

By convexity of ϕ_i, g_i is nonnegative, i.e., $g_i(z) \geq 0$ for all z. Hence, by non-negativity and smoothness g_i is self-bounded (see Section 12.1.3 in [82] or set $z = \hat{z} - \frac{1}{\tilde{L}_i} g_i'(\hat{z})$ in (A.8) and rearrange):

$$\|g_i'(z)\|^2 \leq 2\tilde{L}_i g_i(z), \qquad \forall z. \tag{A.9}$$

Differentiating (A.7) w.r.t. z and combining the result with (A.9), used with $z = x_i^T w$ and $z^* = x_i^T w^*$, gives

$$\|\phi_i'(x_i^T w) - \phi_i'(x_i^T w^*)\|^2 = \|g_i'(x_i^T w)\|^2 \leq 2\tilde{L}_i g_i(x_i^T w). \tag{A.10}$$

Multiplying (A.10) through by $1/(n\tilde{L}_i)$ and summing over $i \in [n]$ shows that

$$\begin{aligned}
&\frac{1}{n}\sum_{i=1}^n \frac{1}{\tilde{L}_i}\|\phi_i'(x_i^T w) - \phi_i'(x_i^T w^*)\|^2 \leq \frac{2}{n}\sum_{i=1}^n g_i(x_i^T w) \\
&= \frac{2}{n}\sum_{i=1}^n \phi_i(x_i^T w) - \phi_i(x_i^T w^*) - \phi_i'(x_i^T w^*)(x_i^T w - x_i^T w^*) \\
&= 2\left(P(w) - \tfrac{\lambda}{2}\|w\|^2 - P(w^*) + \tfrac{\lambda}{2}\|w^*\|^2 - \lambda(w^*)^T(w - w^*)\right) \\
&= 2\left(P(w) - P(w^*) - \tfrac{\lambda}{2}\|w - w^*\|^2\right),
\end{aligned}$$

where we have used the fact that $\nabla P(w^*) = \phi'(x_i^T w^*)x_i + \lambda w^* = 0$. The first inequality follows because $\tilde{L} = \max_i \tilde{L}_i$. □

A.2 Proof of Lemmas 1.3.1 and 1.3.5

Proof of Lemma 1.3.1. In this case it is assumed that every loss function is convex and we set $\gamma = \lambda\tilde{L}$ (1.12). For convenience, define the following quantities:

$$\mathbf{C_1} \stackrel{\text{def}}{=} \frac{\theta}{n}\sum_{i=1}^{n}(u_i^{(t)} - \alpha_i^*)^2 - \tfrac{2\gamma\theta}{\lambda}\nabla P(w^{(t)})^T(w^{(t)} - w^*) \tag{A.11}$$

$$\mathbf{C_2} \stackrel{\text{def}}{=} \sum_{i=1}^{n}\Big(-\tfrac{\theta}{n}(1 - \tfrac{\theta}{p_i}) + \tfrac{\theta^2 v_i \gamma}{n^2\lambda^2 p_i}\Big)(\kappa_i^{(t)})^2 \tag{A.12}$$

Recall that $A^{(t)}$, $B^{(t)}$ and $D^{(t)}$ are defined in (A.1) and (1.13), respectively, and γ is defined in (1.12). Then,

$$\begin{aligned}
\mathbb{E}[D^{(t+1)}|\alpha^{(t)}] - D^{(t)} = \;& \mathbb{E}[A^{(t+1)} - A^{(t)}|\alpha^{(t)}] + \gamma\mathbb{E}[B^{(t+1)} - B^{(t)}|\alpha^{(t)}] \\
\stackrel{(A.2),(A.3)}{=}\; & -\theta A^{(t)} + \tfrac{\theta}{n}\sum_{i=1}^{n}\Big((u_i^{(t)} - \alpha_i^*)^2 - \Big(1 - \tfrac{\theta}{p_i}\Big)(\kappa_i^{(t)})^2\Big) \\
& + \gamma\Big(-\tfrac{2\theta}{\lambda}\nabla P(w^{(t)})^T(w^{(t)} - w^*) + \sum_{i=1}^{n}\frac{\theta^2 v_i}{n^2\lambda^2 p_i}(\kappa_i^{(t)})^2\Big) \\
\stackrel{(A.11),(A.12)}{=}\; & -\theta A^{(t)} + \mathbf{C_1} + \mathbf{C_2}.
\end{aligned} \tag{A.13}$$

Now, by recalling that $\alpha_i^* = \phi_i'(x_i^T w^*)$ and $u_i = \phi_i'(x_i^T w)$ in (A.6), we have,

$$\begin{aligned}
\mathbf{C_1} \stackrel{(A.11)}{=}\; & \frac{\theta}{n}\sum_{i=1}^{n}(u_i^{(t)} - \alpha_i^*)^2 - \tfrac{2\gamma\theta}{\lambda}\nabla P(w^{(t)})^T(w^{(t)} - w^*) \\
\stackrel{(A.6)}{\leq}\; & 2\theta\tilde{L}\Big(P(w^{(t)}) - P(w^*) - \tfrac{\lambda}{2}\|w^{(t)} - w^*\|^2\Big) - \tfrac{2\gamma\theta}{\lambda}\nabla P(w^{(t)})^T(w^{(t)} - w^*) \\
\stackrel{\gamma=\lambda\tilde{L}}{=}\; & -\gamma\theta\|w^{(t)} - w^*\|^2 + 2\theta\tilde{L}\Big(P(w^{(t)}) - P(w^*) - \nabla P(w^{(t)})^T(w^{(t)} - w^*)\Big) \\
\leq\; & -\gamma\theta\|w^{(t)} - w^*\|^2,
\end{aligned} \tag{A.14}$$

where the last inequality follows from convexity of $P(w)$, i.e.,

$$P(w^{(t)}) - P(w^*) \leq \nabla P(w^{(t)})^T(w^{(t)} - w^*).$$

Combining (A.13) and (A.14) gives

$$\mathbb{E}[D^{(t+1)}|\alpha^{(t)}] - D^{(t)} \leq -\theta A^{(t)} - \gamma\theta\|w^{(t)} - w^*\|^2 + \mathbf{C_2} = -\theta D^{(t)} + \mathbf{C_2}.$$

Rearranging gives the result. □

Proof of Lemma 1.3.5. For this result we assume that the average of the loss functions $\frac{1}{n}\sum \phi_i(\cdot)$ is convex. Note that one can define parameters $\bar{\mathbf{C}}_\mathbf{1}$ and $\bar{\mathbf{C}}_\mathbf{2}$ that are analogous to $\mathbf{C_1}$ and $\mathbf{C_2}$ in (A.11) and (A.12) but with γ replaced by $\bar{\gamma}$. Then, the same arguments as those used in (A.13) can be used to show that

$$\mathbb{E}[\bar{D}^{(t+1)}|\alpha^{(t)}] - \bar{D}^{(t)} \leq -\theta A^{(t)} + \bar{\mathbf{C}}_\mathbf{1} + \bar{\mathbf{C}}_\mathbf{2}. \tag{A.15}$$

Now, note that by Lipschitz continuity of $\phi'(\cdot)$ one has

$$(u_i^{(t)} - \alpha_i^*)^2 = \left(\phi_i'(x_i^T w) - \phi_i'(x_i^T w^{(t)})\right)^2 \leq L_i^2\|w^* - w^{(t)}\|^2. \tag{A.16}$$

Further, since the average of the losses is convex, $P(w)$ is strongly convex, so

$$P(w^*) - P(w^{(t)}) \geq \nabla P(w^{(t)})^T(w^* - w^{(t)}) + \tfrac{\lambda}{2}\|w^* - w^{(t)}\|^2 \tag{A.17}$$

and since w^* is the minimizer

$$P(w^t) - P(w^*) \geq \tfrac{\lambda}{2}\|w^{(t)} - w^*\|^2. \tag{A.18}$$

Now, adding (A.17) and (A.18) gives

$$\nabla P(w^{(t)})^T(w^{(t)} - w^*) \geq \lambda\|w^{(t)} - w^*\|^2. \tag{A.19}$$

Therefore,

$$\begin{aligned}\bar{\mathbf{C}}_{\mathbf{1}} = & \frac{\theta}{n}\sum_{i=1}^{n}(u_i^{(t)}-\alpha_i^*)^2 - \frac{2\bar{\gamma}\theta}{\lambda}\nabla P(w^{(t)})^T(w^{(t)}-w^*) \\ \overset{(A.16),(A.19)}{\leq} & \frac{\theta}{n}\sum_{i=1}^{n}L_i^2\|w^{(t)}-w^*\|^2 - 2\bar{\gamma}\theta\|w^{(t)}-w^*\|^2 \\ \overset{(1.26)}{\leq} & -\bar{\gamma}\theta\|w^{(t)}-w^*\|^2. \end{aligned} \tag{A.20}$$

Thus, from (A.15) and (A.20) we have that $\mathbb{E}[D^{(t+1)}|\alpha^{(t)}] - D^{(t)} \leq -\theta D^{(t)} + \bar{\mathbf{C}}_{\mathbf{2}}$, which is the desired result. □

A.3 Proof of Lemma 1.3.2

Proof. This is easy to verify by derive KKT conditions of optimization problem (1.18), which is

$$\begin{cases} -(n\lambda^2\sum_{i\in I_\kappa}\kappa_i^2)(\sum_{i\in I_\kappa}(n\lambda^2+v_i\gamma)p_i^{-1}\kappa_i^2)^{-2}(-(n\lambda^2+v_i\gamma)p_i^{-2}\kappa_i^2)+\mu & =0, \forall i\in I_\kappa \\ \sum_{i\in I_k}p_i & =1 \end{cases}$$

where μ is the Lagrange multiplier.

By comparing the $|I_k|$ equations in the first equality from the KKT conditions above, we have

$$\frac{p_i}{p_j} = \frac{\sqrt{n\lambda^2+v_i\gamma}|\kappa_i|}{\sqrt{n\lambda^2+v_j\gamma}|\kappa_j|}, \text{ for all } i,j\in I_\kappa. \tag{A.21}$$

Considering $\sum_{i\in I_\kappa}p_i = 1$, we show that the optimal probabilities (1.19). (1.20) can be further derived by combine (1.19) and (1.18). □

A.4 Proof of Theorems 1.3.3 and 1.3.6

Proof of Theorem 1.3.3. Note that substituting p^* (where p^* is defined in Lemma 1.3.2) into $\Theta(\kappa, p^*)$ in (1.17) and using the Cauchy-Schwartz inequality, i.e., $(a^T b)^2 \leq \|a\|^2\|b\|^2$, gives

$$\Theta(\kappa, p^*) = \frac{n\lambda^2 \sum_{i\in I_\kappa} \kappa_i^2}{(\sum_{i\in I_\kappa} \sqrt{v_i\gamma + n\lambda^2}|\kappa_i|)^2} = \frac{n\lambda^2 \sum_{i=1}^n \kappa_i^2}{(\sum_{i=1}^n \sqrt{v_i\gamma + n\lambda^2}|\kappa_i|)^2} \geq \frac{n\lambda^2}{\sum_{i=1}^n (v_i\gamma + n\lambda^2)} \overset{(1.24)}{=} \theta^*. \tag{A.22}$$

The above confirms that θ^* in (1.17) is a (constant) global lower bound of $\Theta(\kappa, p^*)$ at every iteration. Thus, using the arguments following Lemma 1.3.1, setting $p^{(t)} = p^*$ (as computed in Lemma 1.3.2) at each iteration gives

$$\mathbb{E}\left[D^{(t+1)}|\alpha^{(t)}\right] \leq (1-\theta^*)D^{(t)}. \tag{A.23}$$

That is, (1.16) used with $\theta \equiv \theta^*$ holds. Because (A.23) holds at every iteration of Algorithm 1.1, one can show that

$$\mathbb{E}\left[D^{(t)}\right] \leq (1-\theta^*)^t C_0 \leq e^{-\theta^* t} C_0, \tag{A.24}$$

where C_0 is defined in (1.21). Now, note that $P(w)$ is $(L+\lambda)$-smooth, i.e., $P(w) - P(w^*) \leq \frac{\lambda+L}{2}\|w - w^*\|^2$, so

$$D^{(t)} = \tfrac{1}{n}\|\alpha^{(t)} - \alpha^*\|^2 + \gamma\|w^{(t)} - w^*\|^2 \geq \gamma\|w^{(t)} - w^*\|^2 \geq \tfrac{2\gamma}{\lambda+L}(P(w^{(t)}) - P(w^*)).$$

This means that we must find T for which

$$\mathbb{E}[P(w^{(T)}) - P(w^*)] \leq \tfrac{\lambda+L}{2\gamma} e^{-\theta^* T} C_0 \leq \epsilon. \tag{A.25}$$

Subsequently, the expression for T in (1.25) is obtained by multiplying through by $e^{\theta^* T}/\epsilon$, taking natural logs, rearranging and noting that

$$\frac{1}{\theta^*} = \frac{\sum_{i=1}^n (v_i\gamma + n\lambda^2)}{n\lambda^2} = n + \frac{\gamma}{n\lambda^2}\sum_{i=1}^n v_i \overset{(1.12)}{=} n + \frac{\tilde{L}}{n\lambda}\sum_{i=1}^n v_i \overset{(1.22)}{=} n + \frac{\tilde{L}Q}{\lambda}.$$

□

Proof of Theorem 1.3.6. Here we assume that the average loss $\frac{1}{n}\sum_{i=1}^n \phi_i(\cdot)$ is convex, but that individual loss functions $\phi_i(\cdot)$ may not be. The proof of this result is almost identical to the proof of Theorem 1.3.3, but with the parameters defined in Section 1.3.2. Similarly to (A.25) we must find T for which

$$\mathbb{E}[P(w^{(T)}) - P(w^*)] \leq \tfrac{\lambda+L}{2\bar{\gamma}} e^{-\theta^* T}\bar{C}_0 \leq \epsilon, \tag{A.26}$$

where $\bar{\gamma} = \frac{1}{n}\sum_{i=1}^n L_i^2$ is defined in (1.26) and $\bar{C}_0$ is defined in (1.28). The expression T in (1.32) is obtained by multiplying through by $e^{\theta^* T}/\epsilon$, taking natural logs, rearranging and noting that

$$\frac{1}{\theta^*} = \frac{\sum_{i=1}^n (v_i\bar{\gamma} + n\lambda^2)}{n\lambda^2} = n + \frac{\bar{\gamma}}{\lambda^2}\Big(\frac{1}{n}\sum_{i=1}^n v_i\Big) \overset{(1.22)}{=} n + \frac{\bar{\gamma}Q}{\lambda^2}.$$

□

A.5 Proof of Corollary 1.3.4

Proof. Recall that w^* denotes the minimizer of (1.1) and $\alpha_i^* = -\phi'(x_i^T w^*)$. Let Assumption 1.4 hold. Then

$$\begin{aligned}
\Big\|\frac{1}{np_i}\kappa_i^{(t)} x_i\Big\|^2 &\overset{(1.14)}{=} \frac{1}{n^2 p_i^2}(\kappa_i^{(t)})^2 v_i \\
&\overset{(1.19)}{=} \frac{1}{n^2}\left(\frac{\sum_{j\in I_\kappa}\sqrt{n\lambda^2 + v_j\gamma}|\kappa_j^{(t)}|}{\sqrt{n\lambda^2+v_i\gamma}|\kappa_i^{(t)}|}\right)^2 (\kappa_i^{(t)})^2 v_i \\
&\overset{(\mathrm{CS})}{\leq} \frac{1}{n^2}\frac{\sum_{j=1}^n (n\lambda^2 + v_j\gamma)\sum_{j=1}^n (\kappa_j^{(t)})^2}{(n\lambda^2+v_i\gamma)(\kappa_i^{(t)})^2}(\kappa_i^{(t)})^2 v_i \\
&= \frac{\sum_{j=1}^n (n\lambda^2+v_j\gamma)}{n^2(n\lambda^2+v_i\gamma)}\|\kappa^{(t)}\|^2 v_i \\
&\overset{(1.22)}{=} \frac{n^2\lambda^2 + \gamma n Q}{n^2(n\lambda^2+v_i\gamma)}\|\kappa^{(t)}\|^2 v_i.
\end{aligned} \tag{A.27}$$

Taking the (conditional) expectation of (A.27) gives

$$
\begin{aligned}
\mathbb{E}\left[\left\|\frac{1}{np_i}\kappa_i^{(t)}x_i\right\|^2\Big|\alpha^{(t-1)}\right] &= \sum_{i=1}^{n} p_i\left(\frac{n^2\lambda^2+\gamma nQ}{n^2(n\lambda^2+v_i\gamma)}\|\kappa^{(t)}\|^2 v_i\right)\\
&\leq \sum_{i=1}^{n}\left(\frac{n^2\lambda^2+\gamma nQ}{n^2(n\lambda^2+v_i\gamma)}\|\kappa^{(t)}\|^2 v_i\right)\\
&\leq \sum_{i=1}^{n}\left(\frac{n^2\lambda^2+\gamma nQ}{n^3\lambda^2}\|\kappa^{(t)}\|^2 v_i\right)\\
&= \left(\frac{n^2\lambda^2+\gamma nQ}{n^2\lambda^2}\|\kappa^{(t)}\|^2\right)\left(\frac{1}{n}\sum_{i=1}^{n} v_i\right)\\
&\overset{(1.22)}{=} Q\left(1+\frac{\gamma Q}{n\lambda^2}\right)\|\kappa^{(t)}\|^2. \qquad \text{(A.28)}
\end{aligned}
$$

Finally

$$
\begin{aligned}
\|\kappa^{(t)}\|^2 = \mathbb{E}\left[\|\kappa^{(t)}\|^2|\alpha^{(t-1)}\right] &= \mathbb{E}\Big[\sum_{i=1}^{n}\left(\alpha_i^{(t)}+\phi_i'(x_i^T w^{(t)})\right)^2|\alpha^{(t-1)}\Big]\\
&= \mathbb{E}\Big[\sum_{i=1}^{n}\left(\alpha_i^{(t)}-\alpha^*-\phi_i'(x_i^T w^*)+\phi_i'(x_i^T w^{(t)})\right)^2|\alpha^{(t-1)}\Big]\\
&\leq 2\mathbb{E}[\|\alpha^{(t)}-\alpha^*\|^2|\alpha^{(t-1)}]+2L\mathbb{E}[\|w^{(t)}-w^*\|^2|\alpha^{(t-1)}].
\end{aligned}
$$

Combining the last step with (A.28) gives the result. □

The proof of Corollary 1.3.7 is essentially identical, but with the notation established in Section 1.3.2, so we omit it for brevity.

A.6 Proof of Theorems 1.5.5 and 1.5.6

Recall that $A^{(t)}$ and $B^{(t)}$ are defined in (A.1). To prove Theorem 1.5.5 we need the following two conditions to hold,

$$
\mathbb{E}_{\hat{S}}\left[A^{(t+1)}-A^{(t)}|\alpha^{(t)}\right] = -\theta A^{(t)}+\frac{\theta}{n}\sum_{i=1}^{n}\left((u_i^{(t)}-\alpha_i^*)^2-\left(1-\tfrac{\theta}{bp_i}\right)(\kappa_i^{(t)})^2\right), \qquad \text{(A.29)}
$$

$$
\mathbb{E}_{\hat{S}}\left[B^{(t+1)}-B^{(t)}|\alpha^{(t)}\right] \leq -\frac{2\theta}{\lambda}\nabla P(w^{(t)})^T(w^{(t)}-w^*)+\sum_{i=1}^{n}\frac{\theta^2 v_i(\kappa_i^{(t)})^2}{n^2\lambda^2 bp_i}. \qquad \text{(A.30)}
$$

Note that $\mathbb{E}_{\hat{S}}\left[A^{(t+1)} - A^{(t)}|\alpha^{(t)}\right] = \sum_{i=1}^{n} bp_i(A^{(t+1)} - A^{(t)})$, and so (A.29) is obtained by using arguments similar to those used in the proof of (A.2). To show (A.30), first we have

$$\begin{aligned} B^{(t+1)} - B^{(t)} &= \|w^{(t+1)} - w^*\|^2 - \|w^{(t)} - w^*\|^2 \\ &= \|w^{(t)} - \sum_{i\in S} \frac{\theta}{n\lambda bp_i}\kappa_i^{(t)} x_i^T - w^*\|^2 - \|w^{(t)} - w^*\|^2 \\ &= -\frac{2\theta}{n\lambda}\sum_{i\in S}\frac{\kappa_i^{(t)}}{bp_i}x_i^T(w^{(t)} - w^*) + \frac{\theta^2}{n^2\lambda^2}\|\sum_{i\in S}\frac{\kappa_i^{(t)}}{bp_i}x_i\|^2. \end{aligned} \tag{A.31}$$

Therefore, we have

$$\begin{aligned} \mathbb{E}_{\hat{S}}\left[B^{(t+1)} - B^{(t)}|\alpha^{(t)}\right] &= \mathbb{E}_{\hat{S}}\Big[-\frac{2\theta}{n\lambda}\sum_{i\in S}\frac{\kappa_i^{(t)}}{bp_i}x_i^T(w^{(t)} - w^*) + \frac{\theta^2}{n^2\lambda^2}\|\sum_{i\in S}\frac{\kappa_i^{(t)}}{bp_i}x_i\|^2|\alpha^{(t)}\Big] \\ &= -\frac{2\theta}{n\lambda}\sum_{i=1}^{n}\kappa_i^{(t)}x_i^T(w^{(t)} - w^*) + \frac{\theta^2}{n^2\lambda^2}\mathbb{E}_{\hat{S}}\|\sum_{i\in S}\frac{\kappa_i^{(t)}}{bp_i}x_i\|^2. \end{aligned} \tag{A.32}$$

Note that from Section 1.5.4 we have

$$\mathbb{E}_{\hat{S}}\|\sum_{i\in S}\frac{\kappa_i^{(t)}}{bp_i}x_i\|^2 \le \sum_{i=1}^{n} bp_i v_i'\left(\frac{\kappa_i^{(i)}}{bp_i}\right)^2 = \sum_{i=1}^{n}\frac{v_i'(\kappa_i^{(t)})^2}{bp_i}, \tag{A.33}$$

where v_i' is defined in (1.36). We can then derive (A.30) by using (A.33) and $\nabla P(w^{(t)}) = \frac{1}{n}\sum_{i=1}^{n}\kappa_i^{(t)}x_i$.

Proof of Theorem 1.5.5. Define

$$\mathbf{C}(\theta, p^{(t)}, \kappa^{(t)}) \stackrel{\text{def}}{=} \sum_{i=1}^{n}\left(-\frac{\theta}{n}\Big(1 - \frac{\theta}{bp_i}\Big) + \frac{\theta^2 v_i'\gamma}{n^2\lambda^2 bp_i}\right)(\kappa_i^{(t)})^2. \tag{A.34}$$

Then

$$\begin{aligned}
\mathbb{E}_{\hat{S}}[D^{(t+1)} - D^{(t)}|\alpha^{(t)}] &= \mathbb{E}_{\hat{S}}[A^{(t+1)} - A^{(t)}|\alpha^{(t)}] + \gamma\mathbb{E}_{\hat{S}}[B^{(t+1)} - B^{(t)}|\alpha^{(t)}] \\
&\overset{(A.29),(A.30)}{\leq} -\theta A^{(t)} + \frac{\theta}{n}\sum_{i=1}^{n}\left(\left(u_i^{(t)} - \alpha_i^*\right)^2 - \left(1 - \frac{\theta}{bp_i}\right)(\kappa_i^{(t)})^2\right) \\
&\quad +\gamma\left(-\frac{2\theta}{\lambda}\nabla P(w^{(t)})^T(w^{(t)} - w^*) + \sum_{i=1}^{n}\frac{\theta^2 v_i'(\kappa_i^{(t)})^2}{n^2\lambda^2 b p_i}\right) \\
&\overset{(A.14)}{\leq} -\theta A^{(t)} - \theta\gamma\|w^{(t)} - w^*\|^2) + \mathbf{C}(\theta, p^{(t)}, \kappa^{(t)}) \\
&= -\theta D^{(t)} + \mathbf{C}(\theta, p^{(t)}, \kappa^{(t)}). \qquad \text{(A.35)}
\end{aligned}$$

We can then derive the optimal probabilities to ensure that $\mathbf{C}(\theta, p^{(t)}, \kappa^{(t)}) \leq 0$, i.e.,

$$\theta \leq \Theta(p^{(t)}, \kappa^{(t)}) := \frac{n\lambda^2 b\sum_{i\in I_{(\kappa^{(t)})}}(\kappa_i^{(t)})^2}{\sum_{i\in I_{\kappa^{(t)}}}(n\lambda^2 + v_i\gamma)(p_i^{(t)})^{-1}(\kappa_i^{(t)})^2} \qquad \text{(A.36)}$$

and then making θ as large as possible. Indeed, to have largest θ we arrive at the same optimal probabilities as in Lemma 1.3.2. Using these optimal probabilities we find a fixed θ^* such that

$$\theta^* \overset{\text{def}}{=} \frac{n\lambda^2 b}{\sum_{i=1}^{n}(n\lambda^2 + v_i\gamma)}. \qquad \text{(A.37)}$$

Furthermore, the complexity result in this mini-batch setting follows: $\mathbb{E}[P(w^t) - P(w^*)] \leq \epsilon$ holds if

$$T \geq \left(\frac{n}{b} + \frac{\tilde{L}Q'}{b\lambda}\right)\log\left(\frac{(\lambda + L)C_0}{\lambda\tilde{L}\epsilon}\right). \qquad \text{(A.38)}$$

□

Proof of Theorem 1.5.6. Define

$$\bar{\mathbf{C}}(\theta, p^{(t)}, \kappa^{(t)}) \overset{\text{def}}{=} \sum_{i=1}^{n}\left(-\frac{\theta}{n}\left(1 - \frac{\theta}{bp_i}\right) + \frac{\theta^2 v_i'\bar{\gamma}}{n^2\lambda^2 b p_i}\right)(\kappa_i^{(t)})^2. \qquad \text{(A.39)}$$

Now

$$\begin{aligned}
\mathbb{E}_{\hat{S}}[\bar{D}^{(t+1)} - \bar{D}^{(t)}|\alpha^{(t)}] &= \mathbb{E}_{\hat{S}}[A^{(t+1)} - A^{(t)}|\alpha^{(t)}] + \bar{\gamma}\mathbb{E}_{\hat{S}}[B^{(t+1)} - B^{(t)}|\alpha^{(t)}] \\
&\overset{(A.29),(A.30)}{\leq} -\theta A^{(t)} + \frac{\theta}{n}\sum_{i=1}^{n}\left(\left(u_i^{(t)} - \alpha_i^*\right)^2 - \left(1 - \frac{\theta}{bp_i}\right)(\kappa_i^{(t)})^2\right) \\
&\quad + \bar{\gamma}\left(-\frac{2\theta}{\lambda}\nabla P(w^{(t)})^T(w^{(t)} - w^*) + \sum_{i=1}^{n}\frac{\theta^2 v_i'(\kappa_i^{(t)})^2}{n^2\lambda^2 bp_i}\right) \\
&\overset{(A.20)}{\leq} -\theta A^{(t)} - \theta\bar{\gamma}\|w^{(t)} - w^*\|^2) + \bar{\mathbf{C}}(\theta, p^{(t)}, \kappa^{(t)}) \\
&= -\theta D^{(t)} + \bar{\mathbf{C}}(\theta, p^{(t)}, \kappa^{(t)}). \qquad \text{(A.40)}
\end{aligned}$$

Similar arguments to those made in the final stages of the proof of Theorem 1.5.6 can be used to show that if T is given by the expression in (1.41) then $\mathbb{E}[P(w^t) - P(w^*)] \leq \epsilon$. $\square$

Appendix B

Proof in Chapter 3

Early Terminated CG Solver on Indefinite System

In this section we provide the proofs of the main Lemmas 3.3.2 and 3.2.4. We start by proving two technical results in the two following Lemmas.

Lemma B.0.1. *If $A \in \mathbb{R}^{n\times n}$ is symmetric and nonsingular, and the nonzero vectors $p_0, \ldots, p_k$ are H_S-conjugatey, then these vectors are linearly independent.*

Proof. Suppose there exist $\{\alpha_i\}_{i=0}^k$, such that $0 = \sum_{i=0}^k \alpha_i p_i$, for any $i_0 \in [k]$, we then have

$$0 = p_{i_0}^T A(\sum_{i=0}^{k} \alpha_i p_i) = \alpha_{i_0} p_{i_0}^T A p_{i_0}. \tag{B.1}$$

Therefore, we have $\alpha_i = 0$ for all $i \in [k]$. □

Lemma B.0.2. *Suppose $A \in \mathbb{R}^{n\times n}$ is symmetric and nonsingular. Let $d_0 \in \mathbb{R}^n$, for given H_S-conjugate basis $p_0, \ldots, p_{n-1}$ of $\mathbb{R}^n$, the sequence generated according to*

$$d_{k+1} = d_k + \alpha_k p_k \tag{B.2}$$

with $\alpha_k = -\frac{(Ad_k - b)^T p_k}{p_k^T A p_k}$ will converges to the unique solution such that $Ad = b$, where $b \in \mathbb{R}^n$.

Proof. Since A is nonsingular, the unique solution of $Ad = b$ is $d^* = A^{-1}b$. Therefore one

would have

$$d^* - d_0 = \sum_{k=0}^{n-1} \hat{\alpha}_k p_k. \tag{B.3}$$

for some nonzero coefficients $\hat{\alpha}_k$. Note also that $d_n = d_0 + \sum_{k=0}^{n} \alpha_k p_k$ from the iterative scheme. We will show that $\hat{\alpha}_k = \alpha_k$. We can easily derive that $p_k^T A(d^* - d_0) = \hat{\alpha}_k p_k^T A p_k$, and then we have

$$p_k^T A d_0 = p_k^T A(d_0 + \sum_{i=2}^{k} \alpha_{i-1} p_{i-1}) = p_k^T A d_k. \tag{B.4}$$

Therefore, we have

$$\hat{\alpha}_k = \frac{p_k^T A(d^* - d_0)}{p_k^T A p_k} = \frac{p_k^T (b - A d_k)}{p_k^T A p_k} = \alpha_k. \tag{B.5}$$

This implies that as long as we can derive a sequence of H_S-conjugate directions, we will be able to find the unique solution of the system $Ad = b$ (with b in the range of A) even when A is not positive definite. □

Proof of Lemma 3.3.2

Consider that $\nabla q(x) = Ax - b$, the stationary point is then obtained by letting $q(x) = Ax - b = 0$. According to the definite of P_k, denote $\sigma = (p_0^T A p_0, p_1^T A p_1, \ldots, p_{k-1}^T A p_{k-1}) \in \mathbb{R}^k$,

$$P_k^T A P_k = \mathrm{diag}(\sigma). \tag{B.6}$$

Therefore $y_i = -\frac{p_i^T p_0}{p_i^T A p_i}, i = 0, \ldots, k-1$. Note also that

$$\begin{aligned} \alpha_i &= -\frac{(Ax_i - b)^T p_i}{p_i^T A p_i} \\ &= -\frac{(A(x_0 + \sum_{j=0}^{i-1} \alpha_j p_j) - b)^T p_i}{p_i^T A p_i} \\ &= -\frac{p_0^T p_i}{p_i^T A p_i} \;=\; y_i. \end{aligned}$$

Proof of Lemma 3.2.4

We first show that $r_k^T p_i = 0$ for all $i \in [k-1]$. Since $\alpha_0 = r_0^T r_0 / (p_0^T B_t p_0)$, it's obvious that

$$r_1^T p_0 = (r_0 - \alpha_0 B_t p_0)^T p_0 = 0.$$

For $i = k-1$, we have

$$r_k^T p_{k-1} = (r_{k-1} - \alpha_{k-1} B_t p_{k-1})^T p_{k-1} = 0.$$

And for $i \in [k-2]$, by noting that p_i and p_{k-1} are conjugate direction, recursively, we have that

$$r_k^T p_i = (r_{k-1} - \alpha_{k-1} B_t p_{k-1})^T p_i = 0.$$

We then show that $\|d_k\|$ is monotonic increasing. In fact, we have

$$\|d_{k+1}\|^2 = \|d_k + \alpha_k p_k\|^2 = \|d_k\|^2 + \alpha_k^2 \|p_k\|^2 + 2\alpha_k p_k^T d_k,$$

and we only need confirm that $p_k^T d_k \geq 0$, for $k = 0, 1, 2, \ldots$.

Note that

$$p_1^T d_1 = (r_1 + \beta_0 p_0)^T (d_0 + \alpha_0 p_0) = \alpha_0 \beta_0 \|p_0\|^2 \geq 0$$

Consequently, we have

$$\begin{aligned} p_k^T d_k &= (r_k + \beta_{k-1} p_{k-1})^T d_k \\ &= r_k^T d_k + \beta_{k-1} p_{k-1}^T (d_{k-1} + \alpha_{k-1} p_{k-1}) \\ &\geq r_k (d_0 + \sum_{i=1}^{k-1} \alpha_i p_i) \\ &\geq 0. \end{aligned}$$

Therefore, we showed that $\|d_{k+1}\|^2 \geq \|d_k\|^2$, and therefore, we have $\|d\| \geq \|d_1\| = \|g_t\|$.

Detailed Description of Datasets and Experiments

We describe the network architectures and training details for the experimental results reported of this chapter in this Section. The implementation is based on CPU and could be easier to build your own solver.

MNIST The MNIST dataset is a set of 28×28 binary handwritten digit images. There

are $60,000$ samples as training dataset and $10,000$ samples as testing dataset. We use $10,000$ samples as validation set and the hypermeters for training the neural networks are chosen such that the lowest validation error is achieved. And we then fix all the hypermeters and use training and validation together and training for a reasonable long time to obtain the final model parameters.

CIFRA-10 and CIFRA-100 The CIFRA-10 and CIFRA-100 datasets are the set of 32×32 RGB images, with 10 and 100 categories, respectively. Both of datasets consist $50,000$ training and $10,000$ testing samples, $5,000$ samples from training samples are abstracted as the validation dataset.

FC1 FC1 is a small fully connected network where we could evaluate the full Hessian and its eigenvalues explicitly. In this setting, we set three layers network with 5 units in the hidden layer. By which, we would get the neural network parameter size as 517.

FC2 FC2 is a small fully connected network with three layers and 50 units at hidden layer. The neural network parameter size is then 51760.

FC3 FC3 is the neural network structure for the full dataset on both MNIST and CIFRA-10, where we use five layers with three hidden layers of 400, 400 and 150 neurons. The overall parameter size for MNIST and CIFRA10 is then around 0.3M and 1.5M.

Algorithm Description

MSGD mini-batch stochastic gradient descent

ASGD accelerate mini-batch stochastic gradient descent (mini-batch stochastic gradient descent with momentum [92])

NSGD noisy mini-batch stochastic gradient descent (mini-batch stochastic gradient descent with noise [59])

Martens-H Martens' Hessian-free method with stochastic Hessian matrix and Levenberg-Marquardt heuristic [51]

Martens-G Martens' Hessian-free method with stochastic Gauss-Newton matrix and Levenberg-Marquardt heuristic [51]

SINNC Inexact Newton-CG method with stochastic Hessian matrix, Algorithm 3.1

SINTR Inexact Trust-Region method with stochastic Hessian matrix, Algorithm 3.2

SINTR+ Inexact Trust-Region method with stochastic Hessian matrix and extra momentum, Algorithm 3.4

Appendix C

Proof in Chapter 4

C.1 Technical Proofs

Before talking about the main results, the following lemma is used in our analysis.

Lemma C.1.1. *(Proposition 5 in [54]) Consider the sample sets $\mathcal{S}_m$ with size m and $\mathcal{S}_n$ with size n such that $\mathcal{S}_m \subset \mathcal{S}_n$. Let w_m is V_m-suboptimal solution of the risk R_m. If assumptions 4.4.2 and 4.4.3 hold, then the following is true:*

$$R_n(w_m) - R_n(w_n^*) \leq V_m + \frac{2(n-m)}{n}(V_{n-m} + V_m) + 2(V_m - V_n) + \frac{c(V_m - V_n)}{2}\|w^*\|^2, \ \ w.h.p. \tag{C.1}$$

If we consider $V_n = \mathcal{O}(\frac{1}{n^\gamma})$ where $\gamma \in [0.5, 1]$, and assume that $n = 2m$ (or $\alpha = 2$), then (C.1) can be written as (w.h.p):

$$R_n(w_m) - R_n(w_n^*) \leq \Big[3 + \Big(1 - \frac{1}{2^\gamma}\Big)\Big(2 + \frac{c}{2}\|w^*\|^2\Big)\Big] V_m. \tag{C.2}$$

C.1.1 Practical stopping criterion

For the risk R_n, the same as [103] we can define the following auxiliary function and vectors:

$$\omega_*(t) = -t - \log(1-t), \qquad 0 \le t < 1. \tag{C.3}$$

$$\tilde{u}_n(\tilde{w}_k) = [\nabla^2 R_n(\tilde{w}_k)]^{-1/2} \nabla R_n(\tilde{w}_k), \tag{C.4}$$

$$\tilde{v}_n(\tilde{w}_k) = [\nabla^2 R_n(\tilde{w}_k)]^{1/2} v_n. \tag{C.5}$$

We can note that $\|\tilde{u}_n(\tilde{w}_k)\| = \sqrt{\nabla R_n(\tilde{w}_k)[\nabla^2 R_n(\tilde{w}_k)]^{-1} \nabla R_n(\tilde{w}_k)}$, which is the exact Newton decrement, and, the norm $\|\tilde{v}_n(\tilde{w}_k)\| = \delta_n(\tilde{w}_k)$ which is the approximation of Newton decrement (and $\tilde{u}_n(\tilde{w}_k) = \tilde{v}_n(\tilde{w}_k)$ in the case when $\epsilon_k = 0$). As a result of Theorem 1 in the study [103], we have:

$$(1-\beta)\|\tilde{u}_n(\tilde{w}_k)\| \le \|\tilde{v}_n(\tilde{w}_k)\| \le (1+\beta)\|\tilde{u}_n(\tilde{w}_k)\|, \tag{C.6}$$

where $\beta \le \frac{1}{20}$. Also, by the equation in (C.5), we know that $\|\tilde{v}_n(\tilde{w}_k)\| = \delta_n(\tilde{w}_k)$.

As it is discussed in the section 9.6.3. of the study [9], we have $\omega_*(t) \le t^2$ for $0 \le t \le 0.68$. According to Theorem 4.1.13 in the study [61], if $\|\tilde{u}_n(\tilde{w}_k)\| < 1$ we have:

$$\omega(\|\tilde{u}_n(\tilde{w}_k)\|) \le R_n(\tilde{w}_k) - R_n(w_n^*) \le \omega_*(\|\tilde{u}_n(\tilde{w}_k)\|). \tag{C.7}$$

Therefore, if $\|\tilde{u}_n(\tilde{w}_k)\| \le 0.68$, we have:

$$\begin{aligned} R_n(\tilde{w}_k) - R_n(w_n^*) &\le \omega_*(\|\tilde{u}_n(\tilde{w}_k)\|) \le \|\tilde{u}_n(\tilde{w}_k)\|^2 \\ &\overset{(C.6)}{\le} \tfrac{1}{(1-\beta)^2}\|\tilde{v}_n(\tilde{w}_k)\|^2 = \tfrac{1}{(1-\beta)^2}\delta_n^2(\tilde{w}_k) \end{aligned} \tag{C.8}$$

Thus, we can note that $\delta_n(\tilde{w}_k) \le (1-\beta)\sqrt{V_n}$ concludes that $R_n(\tilde{w}_k) - R_n(w_n^*) \le V_n$ when $V_n \le 0.68^2$.

C.2 Proof of Theorem 4.4.4

According to the Theorem 1 in [103], we can derive the iteration complexity by starting from w_m as a good warm start, to reach w_n which is V_n-suboptimal solution for the risk R_n. By Corollary 1 in [103], we can note that if we set ϵ_k the same as (4.13), after K_n iterations we reach the solution w_n such that $R_n(w_n) - R_n(w_n^*) \leq V_n$ where

$$K_n = \left\lceil \frac{R_n(w_m) - R_n(w_n^*)}{\frac{1}{2}\omega(1/6)} \right\rceil + \left\lceil \log_2(\frac{2\omega(1/6)}{V_n}) \right\rceil. \tag{C.9}$$

Also, in Algorithm 4.2, before the main loop, 1 communication round is needed, and in every iteration of the main loop in this algorithm, 1 round of communication happens. According to Lemma 4.3.1, we can note that the number of PCG steps needed to reach the approximation of Newton direction with precision ϵ_k is as following:

$$\begin{aligned} C_n(\epsilon_k) &= \left\lceil \sqrt{1 + \tfrac{2\mu_n}{cV_n}} \log_2 \left(\frac{2\sqrt{\frac{cV_n + L}{cV_n}} \|\nabla R_n(\tilde{w}_k)\|}{\epsilon_k} \right) \right\rceil \\ &\overset{(4.13)}{=} \left\lceil \sqrt{1 + \tfrac{2\mu_n}{cV_n}} \log_2 \left(\frac{2(cV_n + L)}{\beta c V_n} \right) \right\rceil. \end{aligned} \tag{C.10}$$

Therefore, in every call of Algorithm 4.2, the number of communication rounds is not larger than $1 + C_n(\epsilon_k)$. Thus, we can note that when we start from w_m, which is V_m-suboptimal solution for the risk R_m, T_n communication rounds are needed, where $T_n \leq K_n(1 + C_n(\epsilon_k))$, to reach the point w_n which is V_n-suboptimal solution of the risk R_n, which follows (4.14). Suppose the initial sample set contains m_0 samples, and consider the set

$$\mathcal{P} = \{m_0, \alpha m_0, \alpha^2 m_0, \ldots, N\},$$

then with high probability with $\mathcal{T}$ rounds of communication, we reach V_N-optimal solution for the whole data set:

$$\mathcal{T} \leq \sum_{i=2}^{|\mathcal{P}|} \left(\left\lceil \frac{R_{\mathcal{P}[i]}(w_{\mathcal{P}[i-1]}) - R_{\mathcal{P}[i]}(w^*_{\mathcal{P}[i]})}{\frac{1}{2}\omega(1/6)} \right\rceil + \left\lceil \log_2(\frac{2\omega(1/6)}{V_{\mathcal{P}[i]}}) \right\rceil \right) \left(1 + \left\lceil \sqrt{1 + \tfrac{2\mu_{\mathcal{P}[i]}}{cV_{\mathcal{P}[i]}}} \log_2 \left(\frac{2(cV_{\mathcal{P}[i]} + L)}{\beta c V_{\mathcal{P}[i]}} \right) \right\rceil \right). \tag{C.11}$$

C.3 Proof of Corollary 4.4.5

The proof of the first part is trivial. According to Lemma C.1.1, we can find the upper bound for $R_n(w_m) - R_n(w_n^*)$, and when $\alpha = 2$, by utilizing the bound (C.2) we have:

$$\begin{aligned} K_n &= \left\lceil \tfrac{R_n(w_m)-R_n(w_n^*)}{\frac{1}{2}\omega(1/6)} \right\rceil + \left\lceil \log_2(\tfrac{2\omega(1/6)}{V_n}) \right\rceil \\ &\overset{(C.2)}{\leq} \underbrace{\left\lceil \tfrac{\left(3+\left(1-\frac{1}{2^\gamma}\right)\left(2+\frac{c}{2}\|w^*\|^2\right)\right)V_m}{\frac{1}{2}\omega(1/6)} \right\rceil + \left\lceil \log_2(\tfrac{2\omega(1/6)}{V_n}) \right\rceil}_{:=\tilde{K}_n}. \end{aligned} \tag{C.12}$$

Therefore, we can notice that when we start from w_m, which is V_m-suboptimal solution for the risk R_m, with high probability with $\tilde{T}_n$ communication rounds, where $\tilde{T}_n \leq \tilde{K}(1 + C_n(\epsilon_k))$, and $C_n(\epsilon_k)$ is defined in (C.10), we reach the point w_n which is V_n-suboptimal solution of the risk R_n, which follows (4.15).

Suppose the initial sample set contains m_0 samples, and consider the set

$$\mathcal{P} = \{m_0, 2m_0, 4m_0, \ldots, N\},$$

then the total rounds of communication, $\tilde{\mathcal{T}}$, to reach V_N-optimal solution for the whole data set is bounded as following:

$$\begin{aligned} \tilde{\mathcal{T}} \leq & \sum_{i=2}^{|\mathcal{P}|} \left(\left\lceil \tfrac{\left(3+\left(1-\frac{1}{2^\gamma}\right)\left(2+\frac{c}{2}\|w^*\|^2\right)\right)V_{\mathcal{P}[i-1]}}{\frac{1}{2}\omega(1/6)} \right\rceil + \left\lceil \log_2(\tfrac{2\omega(1/6)}{V_{\mathcal{P}[i]}}) \right\rceil \right) \left\lceil \left(\left\lceil \sqrt{1+\tfrac{2\mu}{cV_{\mathcal{P}[i]}}} \right\rceil \log_2\left(\tfrac{2(cV_{\mathcal{P}[i]}+L)}{\beta c V_{\mathcal{P}[i]}}\right)\right\rceil \right) \\ \leq & \left(\log_2 \tfrac{N}{m_0} + \left(\tfrac{\left(3+\left(1-\frac{1}{2^\gamma}\right)\left(2+\frac{c}{2}\|w^*\|^2\right)\right)}{\frac{1}{2}\omega(1/6)} \tfrac{1-(\frac{1}{2^\gamma})^{\log_2 \frac{N}{m_0}}}{1-\frac{1}{2^\gamma}} V_{m_0} \right) \right. \\ & \left. + \sum_{i=2}^{|\mathcal{P}|} \left\lceil \log_2(\tfrac{2\omega(1/6)}{V_{\mathcal{P}[i]}}) \right\rceil \right) \left(\left\lceil \sqrt{1+\tfrac{2\mu}{cV_N}} \right\rceil \log_2\left(\tfrac{2}{\beta} + \tfrac{2L}{\beta c}\cdot\tfrac{1}{V_N}\right) \right\rceil \Bigg) \\ \leq & \left(2\log_2 \tfrac{N}{m_0} + \left(\tfrac{\left(3+\left(1-\frac{1}{2^\gamma}\right)\left(2+\frac{c}{2}\|w^*\|^2\right)\right)}{\frac{1}{2}\omega(1/6)} \tfrac{1-(\frac{1}{2^\gamma})^{\log_2 \frac{N}{m_0}}}{1-\frac{1}{2^\gamma}} V_{m_0} \right) \right. \\ & \left. + \log_2 \tfrac{N}{m_0} \log_2(\tfrac{2\omega(1/6)}{V_N}) \right) \left(\left\lceil \sqrt{1+\tfrac{2\mu}{cV_N}} \right\rceil \log_2\left(\tfrac{2}{\beta} + \tfrac{2L}{\beta c}\cdot\tfrac{1}{V_N}\right) \right\rceil \Bigg), \quad \text{w.h.p.} \end{aligned}$$

where $\mu = \max\{\mu_{m_0}, \mu_{\alpha m_0}, \ldots, \mu_N\}$.

C.4 Details Concerning Experimental Section

In this section, we describe our datasets and implementation details. Along the whole Chapter 3, we select four datasets to demonstrate the efficiency of our Algorithm 4.1. Two of them are for convex loss case for a binary classification task using logistic model and the other two are non-convex loss for a multi-labels classification task using convolutional neural networks. The details of the dataset are summarized in Table C.1.

Dataset	**# of samples**	**# of features**	**# of categories**
rcv1	20,242	47,326	2
gisette	7,242	5,000	2
Mnist	60,000	28*28	10
Cifar10	60,000	28*28*3	10

Table C.1: Summary of two binary classification datasets and two multi-labels classification datasets

In terms of non-convex cases, we select two convolutional structure for the demonstration. NaiveCNet is a simple two convolutional layer network for Mnist dataset, and Vgg11 is a relative larger model with 8 convolutional layers. The details of the network architecture is summarized in Table C.2. Note that for vgg11, a batch normalization layer is applied right after each convolutional layer.

C.5 Additional Plots

Besides the plots in Section 4.5, we also experimented different data sets, and the other corresponding settings are described in the main body.

Architecture	NaiveCNet	Vgg11
conv-1	$(5 \times 5 \times 16)$, stride=1	$(3 \times 3 \times 64)$, stride=1
max-pool-1	(2×2), stride=2	(2×2),stride=2
conv- 2	$(5 \times 5 \times 32)$, stride=1	$(3 \times 3 \times 128)$, stride=1
max-pool-2	(2×2), stride=2	(2×2), stride=2
conv- 3		$(3 \times 3 \times 256)$, stride=1
max-pool-3		(2×2), stride=2
conv- 4		$(3 \times 3 \times 256)$
max-pool-4		(2×2), stride=2
conv- 5		$(3 \times 3 \times 512)$, stride = 1
max-pool-5		(2×2), stride=2
conv- 6		$(3 \times 3 \times 512)$, stride = 1
max-pool-6		(2×2), stride=2
conv- 7		$(3 \times 3 \times 512)$, stride = 1
max-pool-7		(2×2), stride=2
conv- 8		$(3 \times 3 \times 512)$, stride = 1
max-pool-8		(2×2), stride=2
fc		512
output	10	10

Table C.2: Summary of two convolutional neural network architecture.

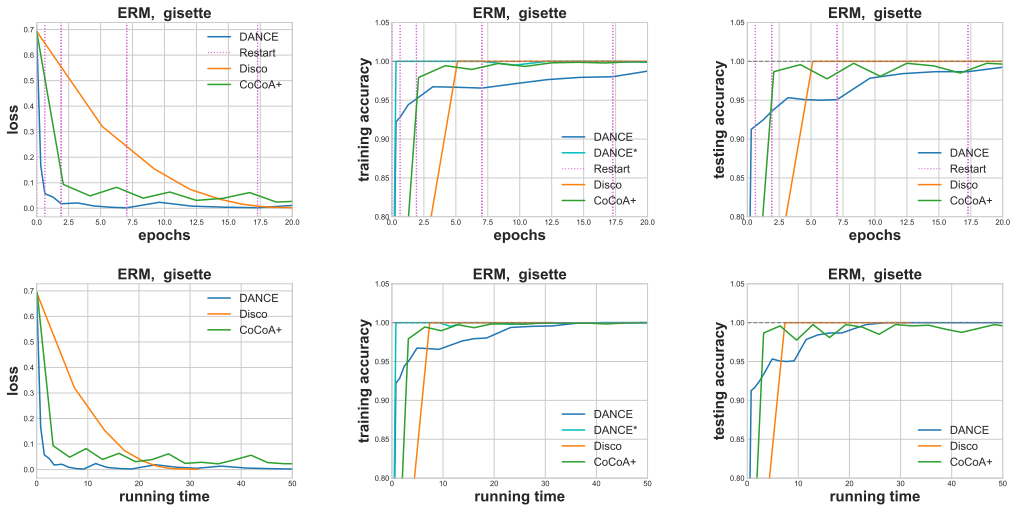


Figure C.1: Performance of different algorithms on a Logistic Regression problem with gisette as dataset. For DANCE algorithm, we set $c = 0.1$ in (4.4), and the regularization parameter is set to be 10^{-4} for other algorithms. For figures in the middle where the y-axis represents training accuracy, the plot *DANCE* is the training accuracy based on the entire training set, while the plot *DANCE** represents the training accuracy based on the current sample size.

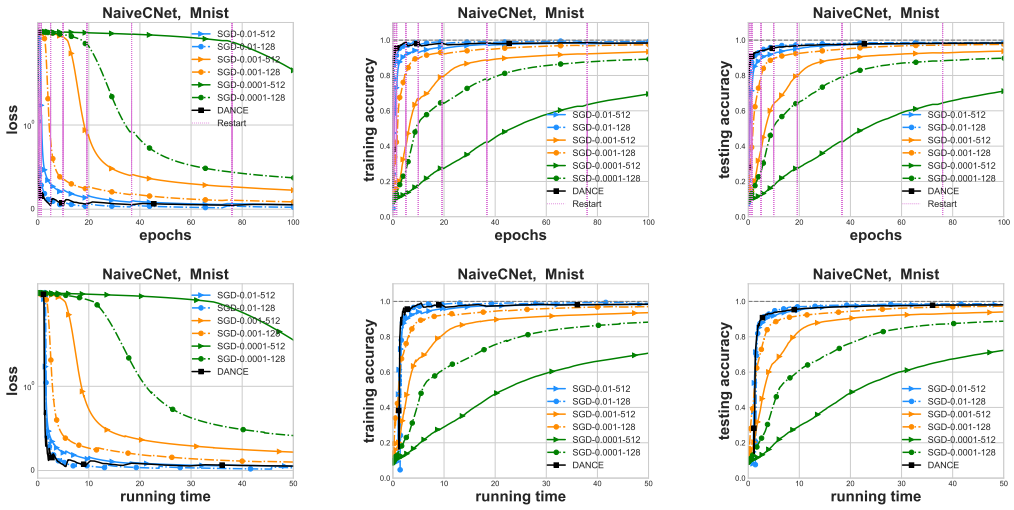


Figure C.2: Comparison between DANCE and SGD with various hyper-parameters on Mnist dataset and NaiveCNet. NaiveCNet is a basic CNN with 2 convolution layers and 2 max-pool layers (see details at Appendix C.4). Figures on the top and bottom show how loss values, training accuracy and test accuracy are changing with respect to epochs and running time. We force two algorithms to restart (double training sample size) after achieving the following number of epochs: $0.075, 0.2, 0.6.1.6, 4.8, 9.6, 18, 36, 72$. For SGD, we varies learning rate from $0.01, 0.001, 0.0001$ and batchsize from $128, 512$. One can observe that SGD is sensitive to hyper-parameter settings, while DANCE has few parameters to tune but still shows competitive performance.

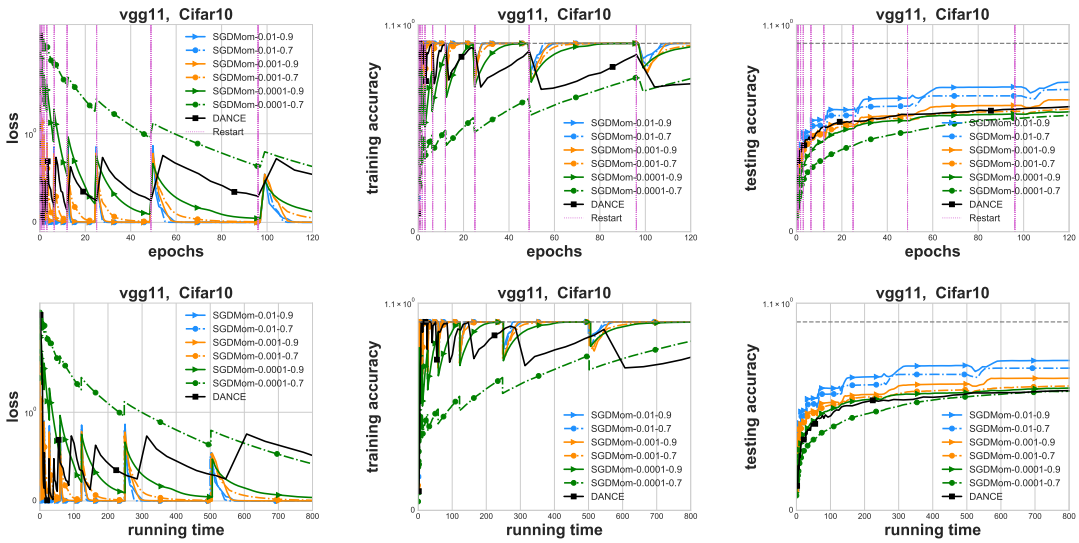


Figure C.3: Comparison between DANCE and with momentum for various hyper-parameters on Cifar10 dataset and vgg11 network. Figures on the top and bottom show how loss values, training accuracy and test accuracy are changing regarding epochs and running time, respectively. We force two algorithms to restart (double training sample size) after running the following number of epochs: $0.2, 0.8, 1.6.3.2, 6.4, 12, 24, 48, 96$. For SGD with momentum, we fix the batchsize to be 256 and varies learning rate from $0.01, 0.001, 0.0001$ and momentum parameter from $0.7, 0.9$. One can observe that SGD with momentum is sensitive to hyper-parameter settings, while DANCE has few hyper-parameters to tune but still shows competitive performance.

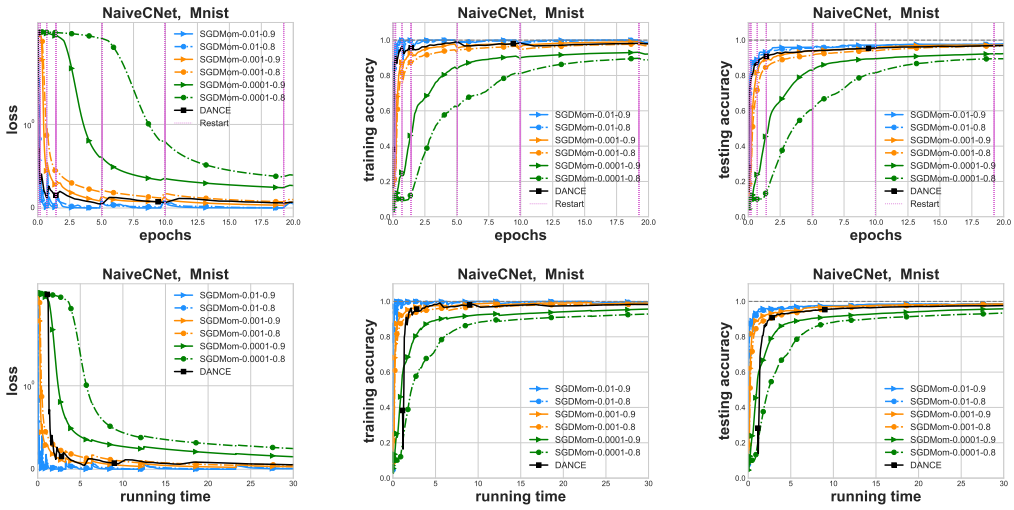


Figure C.4: Comparison between DANCE and SGD with momentum for various hyper-parameters on Mnist dataset and NaiveCNet. Figures on the top and bottom show how loss values, training accuracy and test accuracy are changing regarding epochs and running time, respectively. We force two algorithms to restart (double training sample size) after running the following number of epochs: $0.075, 0.2, 0.6.1.6, 4.8, 9.6, 18, 36, 72$. For SGD with momentum, we fix the batchsize to be 128 and set learning rate to be $0.01, 0.001, 0.0001$ and momentum parameter to be $0.8, 0.9$. One could observe that SGD with momentum is sensitive to hyper-parameter settings, while DANCE has few hyper-parameters to tune but still shows competitive performance.

Appendix D

Notation and Symbols

Basic Objects:

$A, B, \ldots$	matrices
$a, b, \ldots$	vectors
$\alpha, \beta, \ldots$	parameters
$i, j, \ldots$	indices
ξ	random variable
e_i	the unit vector where the i-th element is 1
I_n **or** I	the $n \times n$ identity matrix
$\lambda_i(A)$ **or** λ_i	the i-th (ordered from smallest to largest) eigenvalue of matrix A

Sets:

$\mathbb{R}^n$	the real n-dimensional vector space
$\mathbb{R}^n_+$	the set of nonnegative vectors or $\mathbb{R}^n$
$\mathcal{C}^k$	the set of k-th continuous and differentiable functions in $\mathbb{R}^n$

$[n]$ index set $\{1, 2, \ldots, n\}$

S a sampling subset of $[n]$

Relations:

$A \succ 0$ A is a symmetric positive definite matrix

$A \succeq 0$ A is a symmetric positive semidefinite matrix

$A \prec 0$ A is a symmetric negative definite matrix

$A \not\succeq 0$ A is a symmetric indefinite matrix

Operators, functions:

$\mathbb{E}[\xi]$ expectation of random variable ξ

$\mathbb{P}[X]$ probability of event X

$\nabla f(x)$ **or** $f'(x)$ gradient of function f at point $x \in \mathbb{R}^n$

$\nabla^2 f(x)$ **or** $f''(x)$ Hessian of function f at point $x \in \mathbb{R}^n$

$\frac{\partial f(x)}{\partial x_i}$ partial derivative of $f(x)$ at x_i

$\mathcal{R}_v f(x)$ direction derivative of $f(x)$ along direction v, i.e., $\mathcal{R}_v f(x) = \frac{\partial}{\partial r} f(x + rv)|_{r=0}$

$x^T y$ inner product of x and y

$\|x\|$ L_2-norm of the vector x

$\|A\|$ 2-norm/spectral norm of the matrix A

$|S|$ number of elements in sampling set S, or cardinal number of sampling set S

$\text{span}(x, y, \ldots)$ subspace spanned by vectors $\{x, y, \ldots\}$

Biography

Xi He got his B.S. degree in Mathematics from Nankai University, China, in 2012. He then pursued his M.S. degree in Computational Mathematics from Nankai University in 2014. At the same year, he started his Ph.D. program at Industrial and Systems Engineering department in Lehigh University, USA. He worked with Dr. Martin Takáč in the area of large-scale optimization in machine learning problems. The research focus of him includes first-order stochastic primal-dual optimization for convex/nonconvex empirical risk minimization and distributed second-order stochastic optimization for deep neural networks. He published works to Frontiers in Applied Mathematics and Statistics, AAAI and NIPS workshops. During his Ph.D., he also did internships in Siemens Corporation Research, Alliance Data, and JPMorgan Chase & Co., as a data scientist or quantitative researcher.