Nonconvex Optimization Methods for Large-scale Statistical Machine Learning

by

Rui Zhu

A thesis submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Computer Engineering

Department of Electrical and Computer Engineering

University of Alberta

Abstract

The development of modern machine learning systems not only provides the opportunity of applications in various fields but also creates many challenges. The core of machine learning is to train a model based on a dataset, which can be posed as solving an optimization problem, usually expressed regarding carefully chosen losses and regularizers. However, those formulated optimization problems are more complicated than they appear. Many of them are not convex in model parameters, and we need advanced techniques to solve them. This thesis focuses on how to develop algorithms that can provide solutions for nonconvex optimization problems at different levels.

Some nonconvex optimization problems have specific inner structure, and this allows us to develop convex approximations. In this way, we can turn to find solutions for convex approximation problems. This thesis firstly focuses on an instance of this approach for matrix and tensor estimation problem. Given a subset of observations, i.e., incomplete entries in a matrix (or a tensor), our target here is to find a matrix (or a tensor) that has the minimum rank. We use Schatten-p norm ($1 \leq p \leq 2$) to approximate the matrix rank, a nonconvex function, and propose Iterative Reweighted Schatten-p Norm (IS-p) method to solve this convex approximation. Extensive performance evaluations driven by synthesized data and real-world latency measurements show that our proposed approach achieves better and more robust performance than multiple state-of-the-art matrix completion algorithms in the presence of noise.

Secondly, this thesis studies nonconvex optimization problems by exploiting the convexity of objective function from a local view. In machine learning, many objec-

tive functions have specific inner structures near global optimums, and this allows us to find efficient algorithms to solve. In the second part, we focus on an instance of this approach, matrix factorization for skewed data. Matrix factorization has been proved to be powerful tools in many fields including recommender systems and web latency estimation. However, traditional methods estimate conditional means which may be biased in skewed data settings. We propose Quantile Matrix Factorization (QMF) and Expectile Matrix Factorization (EMF) to overcome such issue. We exploit local strong convexity of global optimums for EMF and shows that we can achieve global optimum under certain conditions in EMF by alternating minimization.

Finally, we study general nonconvex nonsmooth optimization problems that are prevalent in machine learning. Modern applications of machine learning have brought challenges in model complexity and data volume, and the focus of this thesis is to design distributed algorithms that allow us to solve these problems using multiple worker nodes in parallel. We propose asynchronous optimization methods that can enable worker nodes not waiting for others once they complete their tasks, which saves time. We implement the new algorithms on the parameter server system and test on large scale real data sets. We successfully demonstrate that our proposed algorithms can almost achieve linear speedup and accelerate large-scale distributed machine learning.

Preface

The introduction to nonconvex optimization and distributed optimization in Chapter 1 presents concepts and ideas from various of classical textbooks and papers, including [1, 2, 3, 4]. Chapter 2 introduces related work in the literature. The rest of this thesis is grounded on joint work with Di Niu, Zongpeng Li, and other co-authors.

Chapter 3 has been published as R. Zhu, B. Liu, D. Niu, Z. Li, and H. V. Zhao, "Network latency estimation for personal devices: A matrix completion approach," *IEEE/ACM Transactions on Networking (ToN)*, 25(2):724-737, April 2017.

Chapter 4 has been published as R. Zhu, D. Niu, and Z. Li, "Robust Web Service Recommendation via Quantile Matrix Factorization," *in the Proceedings of IEEE INFOCOM 2017*, Atlanta, GA, USA, May 1-4, 2017.

Chapter 5 has been published as R. Zhu, D. Niu, L. Kong, and Z. Li, "Expectile Matrix Factorization for Skewed Data Analysis," *in the Proceedings of AAAI 2017*, San Francisco, CA, USA, February 4-9, 2017.

In Part III, Chapter 6, 7 are joint work with Di Niu and Zongpeng Li [5, 6], which is based on our technical reports [5] and [6]. Chapter 8 is joint work with Di Niu, which is based on our technical report [7].

To my family.

The test of a first-rate intelligence is the ability to hold two opposed ideas in mind at the same time and still retain the ability to function.

– F. Scott Fitzgerald, 1936.

Acknowledgements

My time at the University of Alberta has been a great time in my life. My experiences at the University are part of who I am, and have dramatically shaped my foreseeable future.

At the University of Alberta, I have been very fortunate to be supervised by Dr. Di Niu. I am grateful to them for giving me a change to work with him and providing an excellent research environment with their encouragement, mentoring, wisdom, and kindness. Not only have I acquired invaluable background form his diverse knowledge and intuition in many topics, they have also shown the path to becoming a productive scientist with his brilliant approaches. I am grateful to Dr. Zongpeng Li for his fruitful discussion during the last few years.

I also thank the members of my thesis examining committee, Hao Liang, Hamzeh Khazaei, Ming Zuo, and Liang Zhao, for taking time from their tight schedule to gather for my defense, for their great feedback on this work, as well as the lively discussions.

I would also like to thank Hongwen Zhang and Husam Kinawi for taking me as an intern at Wedge Networks in 2017. I gained valuable industrial research experience.

I also thank my family for their unwavering support. Finally, I thank Yali Yuan for her support and encouragement with love and understanding. She has been my cheerleader, my project manager, my dietitian, my lover, and my friend. Despite the completion of my PhD, I suspect that part of my nonsense is going to persist. My hope is that her company will persist as well.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Many machine learning problems can be formulated as the following optimization problem:

$$\min_{\mathbf{x}\in\mathcal{X}} \quad \Psi(\mathbf{x}) := f(\mathbf{x}) + h(\mathbf{x}), \tag{1.1}$$

where $\mathcal{X} \subseteq \mathbb{R}^d$ is a compact convex set. In the context of machine learning, we call $f(\mathbf{x})$ as the loss function that shows how much cost if we choose x as the parameters of a model, $h(\mathbf{x})$ as the regularizer term that introduces some desired structures on the parameters. Usually, $f(\mathbf{x})$ is in the form of finite-sum like $\frac{1}{n}\sum_{i=1}^{n} f_i(\mathbf{x})$, where n is the size of a dataset and $f_i(\mathbf{x})$ is an individual loss function. Problems like (1.1) are usually referred as *empirical risk minimization* (ERM) problems, and they are prevalent in machine learning, as exemplified by deep learning with regularization [8, 9, 10], LASSO [11], sparse logistic regression [12], robust matrix completion [13, 14], and sparse support vector machine (SVM) [4].

Before proceeding any further, we need to understand the challenges of this problem in the context of modern machine learning applications. The past decades have witnessed a myriad of successes of machine learning applications in self-driving cars, recommender systems, e-commerce systems, autonomous robots, *etc*. Modern machine learning applications, however, have introduced new challenges within the following two categories:

1. **Model complexity.** Traditional machine learning methods focus on developing convex models and algorithms, *e.g.*, SVM and logistic regression. However, nonconvex models like deep learning have achieved exciting progresses. These highly complex models have state-of-the-art performance and are now transforming our world. To train a deep neural network [15] which can include

hundreds of layers with millions of parameters, we need to develop fast and efficient optimization techniques to address model complexity challenges.

2. **Large-scale and distributed data.** With advances in methods like streaming data processing, wireless sensory networks and Internet-of-Things, collecting and managing large volumes of datasets become feasible and popular. Due to the large dataset size, it is distributed across several computing nodes to store, manage, and process. Modern machine learning systems like Parameter Server [16], GraphLab [17] and TensorFlow [18] can handle dataset size in the order of terabytes, and this scale of data brings challenges in heavy burdens on computational and communication workloads.

The main focus of this thesis is to make progresses towards highly complex nonconvex optimization problems addressing these important aspects of the modern machine learning applications. This chapter introduces the background material for this thesis focusing on nonconvex and distributed optimization. Some important notations and terminologies are also introduced in this thesis. Finally, we summarize the contributions of this thesis.

1.1 Nonconvex Optimization

In this section, we will take a brief view of nonconvex optimization. Let us start with recalling some definitions and terminologies that will be used throughout this thesis. A vector $\mathbf{x} = (x_1, \ldots, x_d) \in \mathbb{R}^d$ is a *column* vector by default. We use $A_{i,j}$ (or $\mathbf{A}_{i,j}$) to denote the (i,j)-th entry of the matrix $\mathbf{A}$. We use $\|\cdot\|$ to denote a norm. Popular norms include Frobenius norm $\|\cdot\|_F$ and spectral norm $\|\mathbf{A}\|_2$ when $\mathbf{A}$ is a matrix, and ℓ_p norm $\|\mathbf{x}\|_p$ when $\mathbf{x}$ is a vector[1]. We also denote $\langle \cdot, \cdot \rangle$ as the inner product. For two matrices $\mathbf{A}$ and $\mathbf{B}$ with the same size, their inner product is defined as $\langle \mathbf{A}, \mathbf{B} \rangle = \mathrm{tr}(\mathbf{A}^\top \mathbf{B}) = \sum_{i,j} \mathbf{A}_{i,j} \mathbf{B}_{i,j}$. For a multi-variate function $f(\mathbf{x}) := f(x_1, \ldots, x_d)$, we denote the partial derivative w.r.t. to its i-th coordinate (or block) x_i as $\nabla_i f(\mathbf{x})$.

Some chapters of this thesis will discuss optimization problems on matrices and we will introduce some additional notations here. Let $\mathbf{A}$ be a matrix with the size of $m \times n$ (w.l.o.g., we assume $m \geq n$). The vector of eigenvalues of $\mathbf{A}$ is

[1] A potential ambiguity here is the notation $\|\cdot\|_2$ which may stand for both ℓ_2 norm and spectral norm, since spectral norm is induced by the ℓ_2 norm. In fact, spectral norm is only discussed in Chapter 5 in this thesis.

denoted by $\lambda(\mathbf{A}) = (\lambda_1(\mathbf{A}), \lambda_2(\mathbf{A}), \ldots, \lambda_n(\mathbf{A}))$, and they are ordered as $\lambda_1(\mathbf{A}) \geq \lambda_2(\mathbf{A}) \geq \ldots \geq \lambda_n(\mathbf{A}) \geq 0$. The vector of diagonal elements of $\mathbf{A}$ is denoted by $d(\mathbf{A}) = (d_1(\mathbf{A}), d_2(\mathbf{A}), \ldots, d_n(\mathbf{A}))$. For any $\mathbf{A}$, there exists the singular value decomposition. The singular values are arranged in decreasing order and denoted by $\sigma_1(\mathbf{A}) \geq \sigma_2(\mathbf{A}) \geq \ldots \geq \sigma_n(\mathbf{A}) \geq 0$. Sometimes we also denote $\sigma_{\max}(\mathbf{A})$ as its maximum singular value and $\sigma_{\min}(\mathbf{A})$ as its minimum singular value. It is clear that the singular values of $\mathbf{A}$ are the nonnegative square roots of the eigenvalues of the positive semidefinite matrix $\mathbf{A}^\top\mathbf{A}$, or equivalently, they are the eigenvalues of the positive semidefinite square root $(\mathbf{A}^\top\mathbf{A})^{1/2}$, so that $\sigma_i(\mathbf{A}) = [\lambda_i(\mathbf{A}^\top\mathbf{A})]^{1/2} = \lambda_i[(\mathbf{A}^\top\mathbf{A})^{1/2}], \quad (i = 1, \ldots, n)$.

Definition 1.1.1 (Lipschitz Gradient)**.** *A function f is called L-Lipschitz gradient, if there exists a constant $L > 0$ such that*

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

If f is L-Lipschitz gradient, we can prove that f is also L-smooth, which is a common assumption in the analysis of first-order methods [19, 1, 20]:

$$\begin{cases} f(\mathbf{x}) \geq f(\mathbf{y}) + \langle \nabla f(\mathbf{y}), \mathbf{x} - \mathbf{y} \rangle - \dfrac{L}{2}\|\mathbf{x} - \mathbf{y}\|^2, \\ f(\mathbf{x}) \leq f(\mathbf{y}) + \langle \nabla f(\mathbf{y}), \mathbf{x} - \mathbf{y} \rangle + \dfrac{L}{2}\|\mathbf{x} - \mathbf{y}\|^2, \end{cases}$$

for all $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$. In fact, the value of L controls the curvature of f which prevents situations like rapid changes and breakpoints.

Definition 1.1.2 (Convex Set)**.** *A set $\mathcal{X}$ is called convex,* iff. *for all $\mathbf{x}, \mathbf{y} \in \mathcal{X}$ and $0 \leq \lambda \leq 1$, we have $\lambda\mathbf{x} + (1 - \lambda)\mathbf{y} \in \mathcal{X}$.*

Definition 1.1.3 (Convex Function)**.** *Let $\mathcal{X}$ be a convex set. A function f is called convex if*

$$f\left(\lambda\mathbf{x} + (1 - \lambda)\mathbf{y}\right) \leq \lambda f(\mathbf{x}) + (1 - \lambda) f(\mathbf{y}),$$

for all $\mathbf{x}, \mathbf{y} \in \mathcal{X}$ and λ such that $0 \leq \lambda \leq 1$. If we further have a constant $\mu > 0$ s.t.

$$f(\mathbf{y}) \geq f(\mathbf{x}) + \langle \nabla f(\mathbf{x}), \mathbf{y} - \mathbf{x} \rangle + \frac{\mu}{2}\|\mathbf{x} - \mathbf{y}\|^2,$$

we say f is a μ-strongly convex function.

For a convex function f, we also have (*e.g.*, [1, 20]) $\langle \nabla f(\mathbf{x}), \mathbf{x} - \mathbf{y} \rangle \geq f(\mathbf{x}) - f(\mathbf{y})$ for all $\mathbf{x}, \mathbf{y} \in \mathcal{X}$. If f is μ-strongly convex, we have:

Lemma 1.1.1 (Polyak-Łojasiewicz Inequality)**.** *If f is μ-strongly convex, then it also satisfies the following inequality:*

$$\|\nabla f(\mathbf{x})\|^2 \geq 2\mu(f(\mathbf{x}) - f(\mathbf{x}^*))$$

for all $\mathbf{x} \in \mathcal{X}$, where $\mathbf{x}^*$ is a minimizer of $f(\mathbf{x})$.

Proof. From the definition of μ-strongly convex we have

$$\begin{aligned} f(\mathbf{x}) - f(\mathbf{x}^*) \leq & \langle \nabla f(\mathbf{x}), \mathbf{x} - \mathbf{x}^* \rangle - \frac{\mu}{2}\|\mathbf{x} - \mathbf{x}^*\|^2 \\ = & -\frac{1}{2}\|\sqrt{\mu}(\mathbf{x} - \mathbf{x}^*) - \frac{1}{\sqrt{\mu}}\nabla f(\mathbf{x})\|^2 + \frac{1}{2\mu}\|\nabla f(\mathbf{x})\|_2^2 \\ \leq & \frac{1}{2\mu}\|\nabla f(\mathbf{x})\|^2. \end{aligned}$$

□

Next, we introduce some definitions on the regularizer term $h(\mathbf{x})$ which introduces some inner structures on $\mathbf{x}$. A key difference for h is that, we do not assume smoothness on $h(\mathbf{x})$, and gradient of $h(\mathbf{x})$ may not exist or discontinuous at some points in $\mathcal{X}$. We consider an extension of gradient to handle the nonsmoothness of $h(\mathbf{x})$.

Definition 1.1.4 (Subdifferential e.g., [21])**.** *We say a vector $\mathbf{p} \in \mathbb{R}^d$ is a subgradient of the function $h : \mathbb{R}^d \to \mathbb{R}$ at $\mathbf{x} \in \text{dom } h$, if for all $z \in \text{dom } h$,*

$$h(\mathbf{z}) \geq h(\mathbf{x}) + \langle \mathbf{p}, \mathbf{z} - \mathbf{x} \rangle. \tag{1.2}$$

Moreover, denote the set of all such subgradients at $\mathbf{x}$ by $\partial h(\mathbf{x})$, which is called the subdifferential of h at $\mathbf{x}$.

Now we introduce the concept of critical point for $\Psi(\mathbf{x})$ as follows:

Definition 1.1.5 (Critical point [22])**.** *A point $\mathbf{x} \in \mathbb{R}^d$ is a critical point of $\Psi(\mathbf{x}) = f(\mathbf{x}) + h(\mathbf{x})$,* iff. *$0 \in \nabla f(\mathbf{x}) + \partial h(\mathbf{x})$.*

Nonconvex optimization are significantly more difficult than convex optimization from many aspects. Generally, finding a stationary point in a convex optimization problem is usually sufficient, since any stationary point of a convex function is a local minimum as well as a global minimum [1], which doesn't hold for a nonconvex function. In fact, stationary points of nonconvex functions are more complicated.

Definition 1.1.6. *Given an L-smooth function f, a point $\mathbf{x} \in \mathbb{R}^d$ is called a:*

1. ***stationary point**, if $\nabla f(\mathbf{x}) = 0$;*
2. ***local minimum**, if $\mathbf{x}$ is a stationary point and there exists a neighborhood set $\mathcal{B}(\mathbf{x}) \subseteq \mathbb{R}^d$ such that for all $y \in \mathcal{B}(\mathbf{x})$, we have $f(\mathbf{x}) \leq f(\mathbf{y})$;*
3. ***global minimum**, if $\mathbf{x}$ is a stationary point and for all $\mathbf{y} \in \mathbb{R}^d$, we have $f(\mathbf{x}) \leq f(\mathbf{y})$;*
4. ***saddle point**, if $\mathbf{x}$ is a stationary point but not a local minimum.*

Therefore, a stationary point of a nonconvex function f has three possibilities: local minimum, global minimum, or a saddle point. This means, stationary points of a nonconvex function may not be good solutions, and we need further techniques to check if they are minimum points. Unfortunately, for some nonconvex functions, it is NP-hard to check if a given solution is a local minimum [23], even hard for finding stationary points and checking the optimality.

1.1.1 Convergence Criteria

Convergence analysis is a significant part of designing and analyzing optimization algorithms for the aforementioned problem. In convex optimization, we aim to measure suboptimality based on $f(\mathbf{x}) - f(\mathbf{x}^*)$ or $\|\mathbf{x} - \mathbf{x}^*\|^2$ as convergence criterion, where $\mathbf{x}^*$ is a minimum point. In some nonconvex optimization problems, we can still use these types of convergence criterion, which can lead to convergence to the *global minimum*. Since nonconvex functions usually have complex geometry of landscape, convergence to global minimum is hard to achieve, or with certain conditions. This calls for more realistic convergence criterion for more popular nonconvex problems.

In most part of this thesis, we assume that f is a smooth function with Lipschitz gradient property. It implies that, we can obtain $\nabla f(\mathbf{x})$ for all $\mathbf{x} \in \mathcal{X}$, which is essential to check if $\mathbf{x}$ is a stationary point. Following convergence analysis from [19] and [24], we use $\|\nabla f(\mathbf{x})\|^2$ to check stationarity of point $\mathbf{x}$. In particular, we will consider the following kind of solutions:

Definition 1.1.7 (ϵ-accuracy point, *e.g.*, [25])**.** *We call a point $\mathbf{x} \in \mathcal{X}$ is ϵ-accurate if its gradient is bounded by $\|\nabla f(\mathbf{x})\|^2 \leq \epsilon$.*

The definition of ϵ-accuracy can help us define iteration complexity as follows:

Definition 1.1.8 (Iteration complexity w.r.t. ϵ-accuracy)**.** *If a deterministic algorithm needs at least K steps to achieve ϵ-accuracy, i.e., $\|\nabla f(\mathbf{x}^k)\|^2 \leq \epsilon$ for all $k \geq K$, we then call its iteration complexity as K. For stochastic algorithms, we replace the definition of ϵ-accuracy by its expectations: $\mathbb{E}[\|\nabla f(\mathbf{x}^k)\|^2] \leq \epsilon$ for all $k \geq K$.*

When a nonsmooth regularizer $h(\mathbf{x})$ is involved, we turn to study convergence to *critical points* and *gradient mapping* is the metric we will use for convergence analysis:

$$P(\mathbf{x}, \mathbf{g}, \eta) := \frac{1}{\eta}\left(\mathbf{x} - \mathbf{prox}_{\eta h}(\mathbf{x} - \eta \mathbf{g})\right), \tag{1.3}$$

$$\mathcal{P}(\mathbf{x}) := P(\mathbf{x}, \nabla f(\mathbf{x}), \eta). \tag{1.4}$$

Lemma 1.1.2 (Nonconvex Nonsmooth Convergence [22])**.** *A point $\mathbf{x}$ is a critical point of* (1.1)*,* iff*. $\mathcal{P}(\mathbf{x}) = 0$.*

From this lemma, we can define iteration complexity for nonconvex nonsmooth optimization problem as follows:

Definition 1.1.9 (c.n.t. Iteration complexity w.r.t. ϵ-accuracy)**.** *If a deterministic algorithm needs at least K steps to achieve $\|\mathcal{P}(\mathbf{x})\|^2 \leq \epsilon$, we call its iteration complexity as K. For a stochastic algorithm, the ϵ-accuracy is defined by $\mathbb{E}[\|\mathcal{P}(\mathbf{x})\|^2] \leq \epsilon$.*

1.1.2 Approaches

Although a general nonconvex optimization problem can be very hard, we can still find a number of instances that can have efficient algorithms to solve in feasible time. In this thesis, we consider thre wee typical approaches for nonconvex optimization.

The first approach is to reformulate the nonconvex problem into a convex one, or approximate it. As we have known, convex optimization problems usually have stronger theoretical results that show convergence to global optimality. Thus, the key of this approach becomes to design such transformation and prove the equivalence. One good example here is to compute the minimum eigenvalue of symmetric matrix $\mathbf{A} \in \mathbb{R}^{d\times d}$, which can be formulated as follows:

$$\min_{\mathbf{x}\neq 0} \frac{\mathbf{x}^\top \mathbf{A}\mathbf{x}}{\|\mathbf{x}\|^2}.$$

Garber *et al*. [26] prove that we can solve the following convex optimization problem repeatedly to find the answer:

$$\min_{\mathbf{y}} \frac{1}{2}\mathbf{y}^\top(\mathbf{A} - \lambda \mathbf{I})\mathbf{y} + \mathbf{y}^\top \mathbf{b},$$

for some $\lambda \in \mathbb{R}$ such that $\mathbf{A} - \lambda \mathbf{I} \succ 0$ and $\mathbf{b} \in \mathbb{R}^d$.

Supported by fruitful convex optimization results, solutions found by this type of methods usually have solid performance guarantees. However, designing methods of this approach is problem-specific and requires special techniques, so this only applies in a limited range of nonconvex optimization problems. In Chapter 3, we will see how to use a convex optimization algorithm to find a matrix with minimum rank, which is a complicated nonconvex problem with various applications.

The second approach is to exploit the convexity of objective function from a local view. Recall that if $f(\mathbf{x})$ is a convex function, we have

$$\langle \nabla f(\mathbf{x}), \mathbf{x} - \mathbf{y} \rangle \geq f(\mathbf{x}) - f(\mathbf{y}),$$

which holds for all $\mathbf{x}, \mathbf{y} \in \mathcal{X}$. In addition, if we have

$$\|\nabla f(\mathbf{x})\|^2 \geq \alpha(f(\mathbf{x}) - f(\mathbf{y})),$$

for all $\mathbf{x}, \mathbf{y}$, the objective function $f(\mathbf{x})$ is called *α-strongly convex* function.

Although nonconvex functions may not satisfy these two inequalities, we can relax it to hold around one point $\mathbf{x}^*$. In particular, we have the following definitions:

Definition 1.1.10 (One-point Convexity and Strongly Convexity)**.** *If function $f(\mathbf{x})$ satisfies the following inequality around the solution $\mathbf{x}^*$:*

$$\langle \nabla f(\mathbf{x}), \mathbf{x} - \mathbf{x}^* \rangle \geq f(\mathbf{x}) - f(\mathbf{x}^*) \tag{1.5}$$

for all $x \in \mathcal{X}$, we say $f(\mathbf{x})$ is a one-point convex function. In addition, if there is some $\alpha \geq 0$ such that the following inequality holds:

$$\|\nabla f(\mathbf{x})\|^2 \geq \alpha(f(\mathbf{x}) - f(\mathbf{x}^*)), \tag{1.6}$$

then we say $f(\mathbf{x})$ is a one-point α-strongly convex function.

This category of problems provide some opportunities to achieve global solutions by achieving *local convergence to global minimum*. That is, given some conditions, we can find a proper initialization scheme to ensure the initial point is close to a global minimum. Due to the one-point convexity property, a number of popular algorithms including gradient descent (GD), alternating minimization (AltMin) that will be introduced later can ensure all iterates are close to the global optimum, which leads to convergence. In Chapter 4 and 5, we will discuss matrix factorization problem

that satisfies these conditions, and we will show how our proposed algorithm achieves global optimality under certain condition by exploiting the one-point convexity (or strongly convexity).

If f does not satisfy any of the above properties, the most natural condition we have is its smoothness, which is equivalent with Lipschitz gradient condition. In this case, the best result we can have is to achieve stationary points or local optimal points. For this type of approaches, our goal is to design algorithms that achieve ϵ-accuracy with fewer iterations, and try to avoid saddle points. In Chapter 6 and 7, we will propose our algorithms that achieve stationary points for a number of large-scale machine learning problems.

1.1.3 Algorithms

We now review some important algorithms for nonconvex optimization. The *gradient descent* algorithm is a generic procedure for minimizing functions, both for convex and nonconvex ones. In this algorithm, the objective function $f(\mathbf{x})$ decreases at its negative gradient direction, *i.e.*:

$$\mathbf{x}^{k+1} \leftarrow \mathbf{x}^k - \eta_k \nabla f(\mathbf{x}^k), \tag{1.7}$$

for all iterations $k = 1, 2, \ldots$. It has been shown that if η_k is sufficiently small, *i.e.*, at most $1/L$, we have $f(\mathbf{x}^{k+1}) \leq f(\mathbf{x}^k)$. Therefore, one starts with a point $\mathbf{x}^0 \in \mathbb{R}^d$, and solutions generated by gradient descent are forming a non-increasing sequence:

$$f(\mathbf{x}^0) \geq f(\mathbf{x}^1) \geq \ldots,$$

which leads to convergence. Gradient descent has been proved to converge to stationary points with iteration complexity of $O(1/\epsilon)$ (*e.g.* [20]).

Gradient descent is one of the most well-known optimization algorithms for its effectiveness and simplicity. However, gradient descent has some drawbacks that limits its applications. *Firstly*, gradient descent does not provide guarantee that $\mathbf{x}^k \in \mathcal{X}$ in each iteration, so if constraints exist (*i.e.*, $\mathcal{X} \subset \mathbb{R}^d$), we need an additional step to project $\mathbf{x}^k$ on $\mathcal{X}$. *Secondly*, gradient descent requires a gradient. This is not hard for smooth objective function $f(\mathbf{x})$, but not for nonsmooth functions. For example, if the objective function $\Psi(\mathbf{x})$ has an ℓ_1 regularizer term $h(\mathbf{x}) = \|\mathbf{x}\|_1$, the gradient of $\Psi(\mathbf{x})$ does not exist at origin and we cannot apply gradient descent at this point. *Finally*, evaluating $\nabla f(\mathbf{x})$ can be time consuming in large-scale machine

learning applications. In machine learning, $f(\mathbf{x})$ is usually a function with the sum of n individual ones, so evaluating $\nabla f(\mathbf{x})$ requires to go through the whole dataset of n samples in each iteration, which is time consuming.

To handle the issue brought by $\mathcal{X}$, we can define a projection operator:

$$\Pi_{\mathcal{X}}(\mathbf{x}) := \operatorname*{argmin}_{z \in \mathcal{X}} \frac{1}{2}\|\mathbf{z} - \mathbf{x}\|^2. \tag{1.8}$$

The intuition of projection operator is to find a point $\mathbf{z} in \mathcal{X}$ that is closet with $\mathbf{x}$, and $\Pi_{\mathcal{X}}(\mathbf{x}) = \mathbf{x}$ when $\mathbf{x}$ is in $\mathcal{X}$. Any distance function that well-defined on $\mathcal{X}$ can be used in projection operator by replacing the squared distance above. Given this operator, we can revise the GD by adding a step to project $\mathbf{x}^k$ at iteration k into $\mathcal{X}$, which is called *projected gradient descent* in the literature:

$$\mathbf{x}^{k+1} \leftarrow \Pi_{\mathcal{X}}(\mathbf{x}^k - \eta_k \nabla f(\mathbf{x}^k)). \tag{1.9}$$

When optimizing $\Psi(\mathbf{x})$ or $h(\mathbf{x})$ that may have a some nonsmooth term, a natural idea is to consider *subgradient* when gradient is not available at some point. Suppose we denote $\mathbf{p} \in \partial\Psi(\mathbf{x}^k)$ as the subgradient at iteration k, we can extend GD into *subgradient descent* as follows:

$$\mathbf{x}^{k+1} \leftarrow \mathbf{x}^k - \eta_k \mathbf{p}, \qquad \text{for } k = 1, 2, \ldots$$

The iteration complexity of subgradient descent is proved (*e.g.*, [27]) to be $O(1/\epsilon^2)$ when achieving ϵ-accuracy.

Another popular variant of GD to handle smoothness is *proximal gradient descent*. It utilizes another operator called *proximal operator*:

Definition 1.1.11 (Proximal operator)**.** *The proximal operator* **prox** *of a point* $\mathbf{x} \in \mathbb{R}^d$ *under a proper and closed function* h *with parameter* $\eta > 0$ *is defined as:*

$$\mathbf{prox}_{\eta h}(\mathbf{x}) = \operatorname*{arg\,min}_{\mathbf{y} \in \mathbb{R}^d} \left\{ h(\mathbf{y}) + \frac{1}{2\eta}\|\mathbf{x} - \mathbf{y}\|^2 \right\}. \tag{1.10}$$

And *proximal gradient descent* performs the following iterative updates:

$$\mathbf{x}^{k+1} \leftarrow \mathbf{prox}_{\eta_k h}(\mathbf{x}^k - \eta_k \nabla f(\mathbf{x}^k)),$$

for $k = 1, 2, \ldots$. If we want to incorporate constraints brought by $\mathcal{X}$, we just add an indicator function in $h(\mathbf{x})$:

$$I_{\mathcal{X}}(\mathbf{x}) = \begin{cases} 0, & \text{if } \mathbf{x} \in \mathcal{X} \\ \infty, & \text{otherwise} \end{cases}. \tag{1.11}$$

Finally, we introduce stochastic optimization algorithms for large-scale machine learning problems. A common issue of many machine learning problems is the cost of evaluating $f(\mathbf{x})$ and $\nabla f(\mathbf{x})$, since $f(\mathbf{x})$ consists of calculating functions for a large number of data points. *Stochastic gradient descent* (SGD) attempts to alleviate this problem by taking a subset of data points to calculate a gradient for each iteration, which requires fewer calculations than gradient descent discussed above. In particular, at iteration k, we perform the following equation:

$$\mathbf{x}^{k+1} \leftarrow \mathbf{x}^k - \frac{\eta_k}{|\Xi_k|} \sum_{\xi \in \Xi_k} \nabla F(\mathbf{x}^k; \xi).$$

In SGD, the gradient ∇f is replaced by the gradients from a random subset of training samples, denoted by Ξ_k at iteration k. We call Ξ_k as a *mini-batch*. Since ξ is a random variable indicating a random index in $\{1, \ldots, n\}$, $F(\mathbf{x}; \xi)$ is a random loss function for the random sample ξ, such that $f(\mathbf{x}) := \mathbb{E}_\xi[F(\mathbf{x}; \xi)]$.

SGD is one of the most popular algorithm in modern machine learning problems, including training deep neural networks. Therefore, it is included in all deep learning frameworks (to our best knowledge). We can still guarantee that SGD can converge to stationary points. However, we have to note that SGD has slower convergence rate than GD, since the gradient calculated in each iteration is *noisy* due to random sampling. In fact, to achieve ϵ-accuracy, SGD needs $O(1/\epsilon^2)$ iterations. Note that unlike GD, SGD cannot achieve 0-accuracy by constant learning rate, even we take infinite steps. Therefore, SGD needs learning rate decay schemes in practical applications.

1.2 Parallel and Distributed Optimization

So far we have discussed the challenges of modern machine learning from the perspective of nonconvex optimization. In this section, we focus on how to handle the issue of large scales for modern machine learning. Recent years have witnessed rapid development of parallel and distributed computation frameworks for large-scale machine learning problems. One popular architecture is called *parameter server* [8, 16], which consists of some worker nodes and server nodes, shown in Fig. 1.1. In this architecture, one or multiple master machines play the role of parameter servers, which maintain the model x. All other machines are *worker nodes* that communicate with the server for training machine learning models.

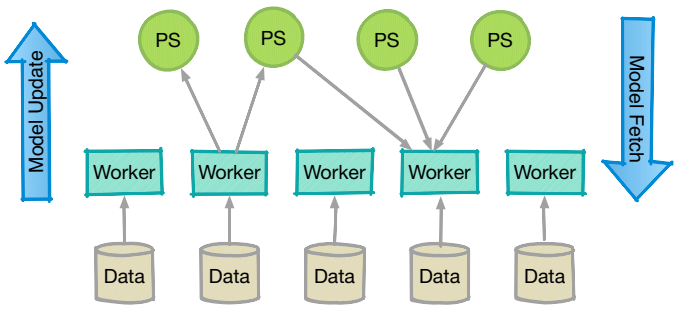


Figure 1.1: Data flow of Parameter Servers. "PS" represents a parameter server task and "Worker" represents a worker task.

A number of machine learning problems can use parameter server architecture to scale up training. Examples include train deep neural networks [8], large scale sparse logistic regression [16], industrial scale factorization machine [28], natural language topic models [29], *etc*. Despite problem specific details, workers perform the following steps in each iteration:

1. Workers "pull" the current model parameters from server nodes.
2. Workers get some data points to calculate model updates.
3. Workers "push" model updates to server nodes.

We can clearly see that "pull" and "push" are two essential primitives to describe communications between worker nodes and server nodes. In the rest of this section, we will discuss some variants under the parameter server framework for distributed machine learning, which will be discussed in Chapter 6, 7 and 8.

Batch vs. Stochastic Training. Recall that in machine learning, $f(\mathbf{x})$ consists of evaluating n individual losses, and the same with $\nabla f(\mathbf{x})$. Traditional algorithms evaluates the gradient explicitly by using all samples in the training dataset. An optimization algorithm runs in this fashion is called *batch update*. A typical batch update algorithm is gradient descent. In the vanilla gradient descent, we perform the following step:

$$\mathbf{x}^{k+1} \leftarrow \mathbf{x}^k - \eta_k \nabla f(\mathbf{x}^k), \tag{1.12}$$

for all iterations $k = 1, 2, \ldots$.

When the dataset size n is large, evaluating the "true" gradient $\nabla f(\mathbf{x}^k)$ is time and memory consuming, and therefore infeasible. A more practical approach is

to sample a few random data points to evaluate a gradient on these data points in each iteration. Compared with batch update, each iteration can be faster and consume less memory in each iterations. However, the gradient becomes noisy due to sampling, and we need more time to converge. In fact, Ghadimi and Lan [24] prove that stochastic gradient descent can only achieve $O(1/\sqrt{K})$-accuracy after K iterations, which is worse than $O(1/K)$ for batch gradient descent.

Synchronous vs. Asynchronous Update. As discussed above, all workers send their updates to servers in each iteration. Due to potential communication latencies and computational heterogeneity among workers, servers may receive updates at different time. Before proceeding any further iterations, servers can wait for all workers to complete the current iteration, and then send the updated model parameters to servers afterwards. Here we need a *synchronization* step to ensure that all workers are in the same iteration, and we call this model update scheme as *synchronous update*. Synchronization here ensures that implementations in parallel mode can behaves exactly the same with sequential ones. Unfortunately, such synchronous updating schemes are often slow since servers have to wait for the slowest participant worker in each iteration.

To circumvent this issue, we can remove this synchronization step and allow workers to update model parameters and continue. This *asynchronous update* scheme works well in practice since it tries to minimize idle time. Existing successful stories include downpour SGD for training deep neural networks [8], topic models [29], *etc*. However, we should be aware of some drawbacks ahead. Since asynchronous updating schemes use staled information during updating, its convergence is usually slower than synchronous schemes.

Data Storage Service vs. Data shards. We now live in the era of big data. We can easily generate petabytes (PBs) of data streams per day in e-commerce websites, face with terabytes of application samples to be scanned, and watching millions of online videos on YouTube. It is quite common to train a model with large data, and this creates challenges of data storage and management for machine learning. At least, we cannot put everything in a single machine.

One popular way is to store the whole dataset in an external distributed file systems, like HDFS, Amazon S3, and Google Cloud Storage. In this approach, each worker accesses to its external storage service, downloads a number of random samples via some provided APIs, and calculates updates for them to send to servers.

Thanks to recent advances in cloud computing, we can conduct experiments on a cluster using virtual machine instances (Amazon EC2) and the external storage service (Amazon S3) from the same vendor (Amazon Web Service), and network latencies between workers and the storage service can be neglected.

Another common solution is to split a large dataset into some smaller ones, each of which will be distributed into a worker machine as its local dataset (or a *data shard*). Due to its ease of implementation and potential privacy-preserving benefits, this data storage method is widely adopted in practical systems with less results of theoretical analysis in the literature, since data distributions among workers can vary a lot.

Model Shards. In the case of large and complex models, we can split the whole model into several shards as well. Each parameter server keeps one model shard, and workers send updates to according server nodes. For example, if we have 4 model shards without overlaps, workers send updates on shard 1 to server 1 only, and similarly for other servers. It has advantages when the model is sparse. For example, in recommender systems, data samples in a worker node may only have information on 10 users out of 1 million, so this worker only have to send updates on these 10 users only instead of to the whole model. Obviously, it saves bandwidth.

Note that if a distributed machine learning scheme divides a dataset or a mini-batch into several parts and let workers evaluate updates in parallel, this corresponds to *data parallelism* (e.g., [30, 31]). If workers update model in parallel by simultaneously updating different model shards, this refers to *model parallelism* (e.g., [32, 33]). Typically, data parallelism is more suitable when $n \gg d$, i.e., large dataset with moderate model size, and model parallelism is more suitable when $d \gg n$.

Consistent vs. Inconsistent. So far we assume that all parameter server nodes and worker nodes are running in a distributed cluster. In this setting, each node is a physical or virtual machine with its own memory. Worker nodes simply send "push" requests and let server nodes update the model. Server nodes, on the other hand, handle all requests in query queues one-by-one, so "pull" requests and "push" requests are hardly conflict. Therefore, we can see that the model pulled by workers is *consistent* with that stored on the server side at some time.

We can simply extend our idea in a shared-memory systems, e.g., a machine with multiple CPU cores or GPU chips. The parameter server architecture still applies here with some variations. Firstly, read and write may conflict, so each processor

may obtain a model that is *inconsistent* with any state of model in the shared memory at any time. Secondly, we prefer model parallelism to data parallelism, since model parallelism can avoid software locks when updating models asynchronously.

1.3 Overview of Thesis & Our Results

This thesis presents a selection of my work on nonconvex optimization methods for machine learning. The content of the thesis corresponds to three nonconvex approaches as we discussed above: convex reformulation or approximation, exploitation on "one-point" convexity, and finding stationary points or local minima. In particular, Chapter 3 shows how to solve rank minimization problem using convex approximation, which is the first approach. Chapter 4 and 5 propose two novel matrix factorization methods for skewed data, which are common in many fields. To analyze the convergence behavior, we exploit the "one-point" convexity around global optimum points and prove local convergence to global optimum. Finally, we study large-scale optimization methods in Chapter 6 to 8 with focus on computational and communication efficiency. In the rest of this section, we provide an overview of these chapters here.

Part I: Convex Approximation

The first part of the thesis focuses on how to use convex optimization as a powerful tool to solve a nonconvex problem, rank minimization for matrix and tensor estimation. Given a number of possibly noisy observations on a low-rank matrix or tensor, our target is to estimate other unknown entries by finding a matrix or tensor with the least rank. However, the rank function is nonconvex, and it is even worse for tensor estimation since computing the rank is NP-hard [34]. This part includes Chapter 3.

Chapter 3: Iterative Reweighted Schatten-p Norm for Matrix and Tensor Estimation

Estimating a low-rank matrix from a few observed entries is a popular problem in many fields, and the problem studied in Chapter 3 is to find a matrix with minimum rank. Since rank is a nonconvex objective function, we can use a convex objective function to approximate it. For example, it has been proved [35, 36] that matrix nuclear norm, the sum of singular values of the matrix, is the closet convex approximation for matrix rank. Based on this observation, various approaches

have been proposed, including using semidefinite programming [36], singular value thresholding (SVT) [37], etc.

However, the gap between matrix rank and nuclear norm for a matrix is still visible. In this chapter, we propose a new class of algorithms, called *Iterative weighted Schatten-p norm minimization* (IS-p), with $1 \leq p \leq 2$, to approximate rank minimization with weighted Schatten-p norm minimization. In each iteration, our algorithm will attempt to solve a convex optimization problem that minimizes "weighted Schatten-p norm", in which Schatten-p norm is a generalization of matrix nuclear norm and matrix Frobenius norm, then adjust the weight such that the weighted sum of Schatten-p norm is close to the matrix rank, which provides better recovery performance.

Later in this chapter, we further extend the proposed IS-p method to tensor approximation. Unlike matrix rank, computing the tensor rank is proved to be NP-hard [34], and it is more complicated to minimize tensor rank directly. We consider a convex approximation of tensor rank [38] by unfolding the tensor into several matrices, and use IS-p to iteratively minimize the reweighted Schatten-p norm of each unfolded matrix.

Part II: "One-point" Convexity

This part of the thesis studies the second approach by exploiting the local curvature for nonconvex optimization. In particular, we study the matrix factorization problem using this approach. We have seen that we can find a good estimation of a matrix with partially observations by rank minimization, and matrix factorization is another line of methods. The name *matrix factorization* is from the fact that finding a matrix with rank at most r is actually to *factorize* the matrix into two tall ones with only r columns. Chapter 4 and 5 are included in this part.

Chapter 4: Quantile Matrix Factorization

In Chapter 4, we propose a novel scheme, *quantile matrix factorization* (QMF), that focuses on applications when observations are highly skewed. We point out that in reality many datasets, e.g. web service response time dataset and video watching dataset, follow highly skewed distributions. In this case, the conditional mean estimates generated by traditional matrix completion approaches can be heavily biased towards a few outliers, leading to poor web service recommendation performance.

In this chapter, we propose the Quantile Matrix Factorization (QMF) technique

for matrix completion by introducing quantile regression into the matrix factorization framework. We propose a novel and efficient algorithm based on Iterative Reweighted Least Squares (IRLS) to solve the QMF problem involving a nonsmooth objective function. Extensive evaluation based on a large-scale QoS dataset has shown that our schemes significantly outperform various state-of-the-art web service QoS estimation schemes in terms of personalized recommendation performance.

Chapter 5: Expectile Matrix Factorization

This chapter introduces another method of matrix factorization for skewed data. In this chapter, we propose expectile matrix factorization by introducing asymmetric least squares, a key concept in expectile regression analysis, into the matrix factorization framework. We propose an efficient algorithm to solve the new problem based on alternating minimization and quadratic programming. We prove that our algorithm converges to a global optimum and exactly recovers the true underlying low-rank matrices when noise is zero. For synthetic data with skewed noise and a real-world dataset containing web service response times, the proposed scheme achieves lower recovery errors than the existing matrix factorization method based on least squares in a wide range of settings.

Part III: Large Scale Optimization Methods

In the third part, we look into asynchronous and distributed optimization problems. In particular, we focus on problems with large scale and parallelism is important. Communication cost is usually the dominating factor when deploying parallel algorithms in a distributed cluster. In order to lower communication cost in parallel algorithms, the distribution optimization algorithm needs to be flexible to various node synchronization schemes, which is our focus in this part. This part of the thesis is to design novel asynchronous algorithms for nonconvex nonsmooth problems in various settings, and includes Chapter 6, 7 and 8.

Chapter 6: Asynchronous Blockwise ADMM

A number of algorithms have been proposed to solve nonconvex nonsmooth optimization problem (1.1), and alternating direction multiplier method (ADMM) is one of the most popular algorithm [2]. In ADMM, workers have their own datasets and maintain their own solutions, called *local models*, while the server side maintains a global solution, so workers perform most of the computation. The communication

among workers and the server is just to ensure that local models are *consensus* with the global model, which eliminates a lot of unnecessary communications. Due to its flexibility and communication efficiency, many recent efforts have been made to develop asynchronous distributed ADMM to handle large amounts of training data.

However, all existing asynchronous distributed ADMM methods are based on full model updates and require locking all global model parameters to handle concurrency, which essentially serializes the updates from different workers. In this paper, we present a novel block-wise, asynchronous and distributed ADMM algorithm, which allows different blocks of model parameters to be updated in parallel. The block-wise algorithm may greatly speedup sparse optimization problems, a common scenario in reality, in which most model updates only modify a subset of all decision variables. We theoretically prove the convergence of our proposed algorithm to stationary points for nonconvex general form consensus problems with possibly nonsmooth regularizers.

Chapter 7: Asynchronous Proximal Stochastic Gradient Descent

Many optimization algorithms are designed for small-scale problems, and *batch* gradient is needed. For example, in gradient descent, we should go through all n samples in each iteration, and similarly in our proposed asynchronous ADMM algorithm in Chapter 6. However, in large-scale setting, batch methods become intractable even we split them into several parts. In such scenarios, stochastic methods are preferred. In Chapter 7 and 8, we focus on proximal stochastic gradient descent methods for two parallelism settings.

In Chapter 7, we propose and analyze asynchronous parallel stochastic proximal gradient (Asyn-ProxSGD) methods for nonconvex problems. We consider *data parallelism* in this chapter, in which all workers compute sample gradients for a whole minibatch in parallel. For example, if we have a minibatch with 128 samples, and we have 8 workers, then each worker should compute gradients on 16 samples. We then establish an ergodic convergence rate of $O(1/\sqrt{K})$ for the proposed Asyn-ProxSGD, where K is the number of updates made on the model, matching the convergence rate currently known for AsynSGD (for smooth problems). To our knowledge, this is the first work that provides convergence rates of asynchronous parallel ProxSGD algorithms for nonconvex problems. Furthermore, our results are also the first to show the convergence of any stochastic proximal methods without assuming an increasing batch size or the use of additional variance reduction techniques.

Chapter 8: Asynchronous Block Proximal Stochastic Gradient

This chapter also focuses on asynchronous proximal stochastic gradient method for *model parallelism*. In model parallelism, each worker computes gradients for the whole minibatch but only updates *one block* of the model. For example, suppose we still have 128 samples in a minibatch with 8 workers. Each worker still has to compute gradients on all 128 samples, but it only updates one block instead of the whole model. In contrast, the algorithm discussed in Chapter 7 assumes that all workers should update *the whole model* in each iteration. The motivation here is that large models are prevalent in modern machine learning scenarios, including deep learning, recommender systems, etc., which can have millions or even billions of parameters.

In this chapter, we propose a model parallel proximal stochastic gradient algorithm, AsyB-ProxSGD, to deal with large models using model parallel blockwise updates while in the meantime handling a large amount of training data using proximal stochastic gradient descent (ProxSGD). We prove that AsyB-ProxSGD achieves a convergence rate of $O(1/\sqrt{K})$ to stationary points for nonconvex problems under *constant* minibatch sizes, where K is the total number of block updates. This rate matches the best-known rates of convergence for a wide range of gradient-like algorithms. Furthermore, we show that when the number of workers is bounded by $O(K^{1/4})$, we can expect AsyB-ProxSGD to achieve linear speedup as the number of workers increases.

1.3.1 Bibliographic Notes

My research in this thesis is based on joint works with many co-authors as follows. In Part I, Chapter 3 is based on joint work with Bang Liu, Di Niu, Zongpeng Li and Hong V. Zhao, which is based on our published work in TON [39]. In Part II, Chapter 4 and 5 are joint works with Di Niu, Linglong Kong and Zongpeng Li, which are based on our published work in IEEE INFOCOM [40] and AAAI [41], respectively. In Part III, Chapter 6, 7 and 8 are joint work with Di Niu and Zongpeng Li [5, 6], which is based on our technical report [5], our work submitted to AAAI, and our work submitted to AISTATS [7], respectively.

Chapter 2

Related Work

2.1 Matrix Completion via Rank Minimization

Matrix estimation has wide applications in many fields such as recommendation systems [42], network latency estimation [43], computer vision [44], system identification [45], etc. For example, a movie recommendation system aims to recover all user-movie preferences based on the ratings between some user-movie pairs [42, 46], or based on implicit feedback, e.g., watching times/frequencies, that are logged for some users on some movies [47, 48]. In network or web service latency estimation [43, 49, 50], given partially collected latency measurements between some nodes that are possibly contaminated by noise, the goal is to recover the underlying low-rank latency matrix, which is present due to network path and function correlations.

Given partial and possibly noisy observations on some entries, our target is to find a low-rank matrix that recovers all unknown entries, which is called *matrix completion* in the literature. There are two different approaches to solve this problem. The first approach is to find a matrix with the minimum rank under conditions from observations. Specifically, this approach attempts to solve the following problem:

$$\begin{aligned} \underset{\hat{\mathbf{M}} \in \mathbb{R}^{m \times n}}{\text{minimize}} \quad & \text{rank}(\hat{\mathbf{M}}) \\ \text{subject to} \quad & \hat{\mathbf{M}}_{i,j} = \mathbf{M}_{i,j},\ (i,j) \in \Omega, \end{aligned} \tag{2.1}$$

where Ω is the set of observed entries. Generally, this is an NP-hard problem, and we turn to a more tractable solution by minimizing the nuclear norm, which has been proved to be the closest convex relaxation of matrix rank [35]. Using nuclear norm as a relaxation has a nice connection with compressive sensing [51, 52]: the nuclear norm of a matrix is the sum of its singular values, which can be viewed as the ℓ_1-norm of the vector consisting of all its singular values. The matrix rank can

be viewed as the ℓ_0-norm of that vector, since it is the number of nonzero singular values of a matrix.

A number of algorithms have been proposed to solve (2.1). In [36], Recht *et al.* propose to reformulate (2.1) as a semidefinite program (SDP) and solved to global optima by using standard SDP solvers. This works good for small matrices. To deal with larger matrices, some first order algorithms have been proposed, e.g. the singular value thresholding (SVT) algorithm [37] and its variants [53].

A number of iterative reweighted approximations to rank minimization have also been proposed. The key idea for these methods is to put some weights on singular values such that the weighted matrix norm, for example weighted nuclear norm or weighted Frobenius norm, is close to the matrix rank. Depending on different norms considered, they may differ in recovery performance as well as the formulation of the weight matrix. Daubechies *et al.* propose Iterative Reweighted Least Squares (IRLS-p and sIRLS-p), which minimizes a weighted Frobenius norm in each iteration. Another reweighted algorithm is proposed by Mohan and Fazel [54], which minimizes the weighted nuclear norm in each iteration. In Chapter 3, we will propose IS-p, a family of Iterative reweighted Schatten-p norm method that generalizes these two algorithms, providing better performance in recovery and flexibility to trade off recovery accuracy and running time.

2.2 Matrix Factorization

Another approach for matrix estimation is matrix factorization. In the matrix factorization approach, the rank of the truth matrix $\mathbf{M}^*$ is assumed to be at most r, so we can find two tall matrices $\mathbf{X} \in \mathbb{R}^{m\times r}$ and $\mathbf{Y} \in \mathbb{R}^{n\times r}$ such that $\mathbf{M}^* = \mathbf{X}\mathbf{Y}^\top$. So if we attempt to recover some missing entries in $\mathbf{M}^*$, we solve the following optimization problem:

$$\min_{\mathbf{X}\in\mathbb{R}^{m\times r},\mathbf{Y}\in\mathbb{R}^{n\times r}} \sum_{(i,j)\in\Omega} \mathcal{L}(\mathbf{M}_{i,j}, \langle \mathbf{x}_i, \mathbf{y}_j\rangle). \tag{2.2}$$

Here $\mathbf{x}_i$ denotes the i-th row of $\mathbf{X}$, and same for $\mathbf{y}_j$. The function $\mathcal{L}$ is the loss function that measures how much difference between the observation $\mathbf{M}_{i,j}$ and the estimation $\langle \mathbf{x}_i, \mathbf{y}_j\rangle$. In many cases, we may put additional regularizer terms like $\lambda(\|\mathbf{X}\|_F^2 + \|\mathbf{Y}\|_F^2)$ that penalize solutions with large norm. In fact, by setting $\mathcal{L}$ as the squared error, we can easily obtain this formulation by MAP (maximum a

posteriori) estimation under a certain probabilistic model [55]. Since minimizing (mean) squared error is to estimate the conditional mean, this approach may not work for problems on skewed data. In Chapter 4 and 5, we will see how to choose better loss functions for skewed data distribution.

Although we often choose a convex function $\mathcal{L}$ in (2.2), and the regularizer term is also often convex, the overall problem is unfortunately nonconvex. The reason is that if we find a pair of solutions $\mathbf{X}\mathbf{Y}^\top = \mathbf{M}^*$, we can generate another pair of solutions by letting $\mathbf{X}' = \mathbf{X}\mathbf{P}$ and $\mathbf{Y}' = \mathbf{Y}\mathbf{P}^{-1}$ for any invertible matrix $\mathbf{P} \in \mathbb{R}^{r\times r}$. This also brings challenges to find a proper distance function that measures how far the current iteration apart from the global optimum.

We firstly note that various existing algorithms can find solutions that converge to stationary points, such as alternating minimization (AltMin), gradient descent (GD), and stochastic gradient descent (SGD). AltMin is widely used in matrix factorization [42]. It works by fixing one variable, saying $\mathbf{X}$, and find optimal solutions given $\mathbf{X}$, then fix $\mathbf{Y}$ and do the same procedure again, and iterates until stops. We can see that when fixing either $\mathbf{X}$ or $\mathbf{Y}$, the objective function becomes convex, and thus minimization becomes easy. In fact, we can even find close-form equations for some popular $\mathcal{L}$ like the squared loss. Another benefit for AltMin is that when fixing either $\mathbf{X}$ or $\mathbf{Y}$, we can find solutions for each row in parallel, which makes AltMin very popular for large-scale matrix factorization tasks. Other algorithms like GD and SGD are also popular for large-scale matrix factorization and have been implemented in industrial systems, *e.g.* [17, 28].

Despite its success in applications, theoretical understanding of matrix factorization is rather limited. A lot of work on the local convergence analysis have shown that various algorithms can converge to global optimum given a sufficiently good initialization. In particular, Sun and Luo [14] and Zheng and Lafferty [56] have shown that under some conditions, standard matrix factorization algorithms like AltMin, GD and SGD can exactly recover the true low-rank matrix. They accomplish this by showing geometric property and exploiting the "one-convex" property around global optimum points. Recent works (*e.g.* [57, 58, 59, 60]) provide advanced analyses of matrix completion by more complicated initialization procedure and refined assumptions. In Chapter 5, we will provide our local convergence analysis for our proposed matrix factorization scheme (expectile matrix factorization, EMF) and show that our algorithm can converge to global optimum when observations are

noiseless, which implies exactly recovery of the true low-rank matrix $\mathbf{M}^*$.

2.3 Large Scale Optimization Algorithms

In this section we review several types of optimization algorithms for large scale machine learning problems. We focus on two types: *stochastic optimization* algorithms that may involve randomly choosing samples or coordinates during iterations, and *parallel optimization* algorithms that run on distributed clusters. This section summarizes related work for Chapter 6, 7 and 8.

Stochastic optimization. Stochastic optimization problems have been studied since the seminal work in 1951 [61], in which a classical stochastic approximation algorithm is proposed for solving a class of strongly convex problems. Since then, a series of studies on stochastic programming have focused on convex problems using SGD [62, 63, 64]. The convergence rates of SGD for convex and strongly convex problems are known to be $O(1/\sqrt{K})$ and $O(1/K)$, respectively. For nonconvex optimization problems using SGD, Ghadimi and Lan [24] proved an ergodic convergence rate of $O(1/\sqrt{K})$, which is consistent with the convergence rate of SGD for convex problems.

Stochastic Proximal Methods. When $h(\cdot)$ in (1.1) is not necessarily smooth, there are other methods to handle the nonsmoothness. One approach is closely related to mirror descent stochastic approximation, e.g., [65, 66]. Another approach is based on proximal operators [21], and is often referred to as the *proximal stochastic gradient descent* (ProxSGD) method. Duchi et al. [67] prove that under a diminishing learning rate $\eta_k = 1/(\mu k)$ for μ-strongly convex objective functions, ProxSGD can achieve a convergence rate of $O(1/\mu K)$. For a nonconvex problem like (1.1), rather limited studies on ProxSGD exist so far. Ghadimi *et al.* [68] propose a stochastic proximal gradient (ProxSGD) algorithm and their convergence analysis shows that ProxSGD can converge to stationary points when the objective function is convex. For the nonconvex case, their convergence result relies on the assumption that the minibatch size in each iteration should increase, which may not be practical in implementations. In Chapter 7, we will fill this gap by proving that constant minibatch size is sufficient for convergence to critical points.

Synchronous Parallel Methods. In parallel algorithms, multiple workers are working together to execute one computational task, and the scheme to keep them on the same pace is called *synchronization* in the literature. In the context of

machine learning, synchronous methods ask worker nodes to fetch the model stored on parameter server(s) and the parameter server should *synchronize* all the received updates (*e.g.* stochastic gradients from workers), then *aggregate* them and finally *update* the model.

Here we list some important synchronous parallel algorithms involved in this thesis. Most of them are trying to parallelize their sequential version, which are designed to run in a single machine. Algorithms for matrix factorization like gradient descent and alternating minimization are easy to parallelize by finding solutions for each row of $\mathbf{X}$ (or $\mathbf{Y}$). Another important algorithm that runs in parallel is alternating direction multiplier method (ADMM) [2]. In this algorithm, each worker computes a local model $\mathbf{x}_i$ (in parallel) by minimizing its own objective function f_i, and try to be consensus with the global model $\mathbf{z}$ stored in the parameter server.

Block Coordinate Methods. One special type of parallel methods is the type of block coordinate methods. In these methods, each worker updates one block in parallel, instead of the whole model. Block coordinate methods for smooth problems with separable, convex constraints [69] and general nonsmooth regularizers [70, 71, 72] are proposed. However, the study on *stochastic* coordinate descent is limited and existing work like [73] focuses on convex problems. Xu and Yin [74] study block stochastic proximal methods for nonconvex problems. However, they only analyze convergence to stationary points assuming an increasing minibatch size, and the convergence rate is not provided.

Asynchronous Parallel Methods. To deal with big data, asynchronous parallel optimization algorithms have been heavily studied, although pioneer studies started from 1980s [3]. The rapid development of hardware resources and system designs, recent years have witnessed a lot of successes of asynchronous parallelism in wide range of applications. A number of works focuses on stochastic gradient descent for smooth optimization, *e.g.*, [32, 30, 75, 33, 76] and deterministic ADMM, e.g. [77, 78].

Extending consensus ADMM to the asynchronous setting has been discussed in recent years. In [77], they consider a asynchronous consensus ADMM with assumption on bounded delay, but only convex cases are analyzed. References [79, 80] propose a asynchronous ADMM algorithm with analysis on both convex and nonconvex cases. However, they require each worker solve a subproblem exactly and it might be time costly in some cases. Reference [78] proposes another asynchronous

algorithm that each worker only calculates gradients and updates of all $\mathbf{x}_i, \mathbf{y}_i$ and $\mathbf{z}$ are done on the server side, which can consume a lot of memory in a large cluster. References [81, 82] consider asynchronous ADMM for decentralized networks. In particular, reference [81] assumes that communication links between nodes can fail randomly and provides convergence analysis in a probability-one sense.

To our best knowledge, existing asynchronous ADMM algorithms do not support blockwise updates. In Chapter 6, we make the first attempt to study blockwise updates of ADMM in asynchronous settings.

Asynchronous Parallel Stochastic Methods. Asynchronous algorithms for stochastic gradient algorithms have been discussed and we have known more results for convex objective functions. Agarwal and Duchi [30] provides convergence analysis of asynchronous SGD for *convex smooth* functions. Feyzmahdavian *et al.* [83] extends their analysis for problems with the "convex loss + nonsmooth convex regularizer" form. Niu *et al.* [32] propose a lock free asynchronous implementation on shared memory systems, in which each worker can update a block of model without software locks. They provide convergence analysis for *strongly* convex smooth objectives.

However, theoretical analysis for nonconvex objectives are limited in the literature. Lian *et al.* [75] provide convergence analysis of asynchronous SGD for *nonconvex* smooth objectives. A non-stochastic asynchronous ProxSGD algorithm is presented by [84], which however did not provide convergence rates for nonconvex problems. In Chapter 7, we fill the gap in the literature by providing convergence rates for ProxSGD under constant batch sizes without variance reduction.

Davis *et al.* [85] present a stochastic block coordinate method, which is the closest one with our work in this paper. However, the algorithm studied in [85] depends on the use of a noise term with diminishing variance to guarantee convergence. In Chapter 8, we propose AsyB-ProxSGD and our convergence results of ProxSGD do not rely on the assumption of increasing batch sizes, variance reduction or the use of additional noise terms.

Chapter 3

Iterative Reweighted Schatten-p Norm for Matrix and Tensor Estimation

3.1 Background

In this chapter, we present our first demonstration on the task of Matrix Estimation and its extension to Tensor Estimation. In these problems, a low-rank matrix $\mathbf{M}^* \in \mathbb{R}^{m \times n}$ or a linear mapping $\mathcal{A}(\mathbf{M}^*)$ from the low-rank matrix $\mathbf{M}^*$ is assumed to underlie some possibly noisy observations, where $\mathcal{A} : \mathbb{R}^{m \times n} \to \mathbb{R}^p$. The objective is to recover the underlying low-rank matrix based on partial observations b_i, $i = 1, \ldots, p$. This is also called *matrix sensing* problem in the literature. In particular, if the linear mapping $\mathcal{A}$ is actually a projection operator, which takes one entry of $\mathbf{M}^*$ and vanishes all others to zero, observations b_i's form an incomplete matrix, and the task to estimate the original $\mathbf{M}^*$ is called *matrix completion*.

To exploit the low-rank structure, there are two approaches in the literature that formulates such problem. One is to find a matrix with minimum rank, which is referred as *rank minimization approach* in the literature. The other approach is to find a matrix that has rank at most r, which is a prior given hyperparameter. Since the matrix $\mathbf{M}^*$ with the rank of r can be represented as the product of two factor matrices, i.e., $\mathbf{M}^* = \mathbf{U}\mathbf{V}^\top$, where $\mathbf{U} \in \mathbb{R}^{m \times r}$ and $\mathbf{V} \in \mathbb{R}^{n \times r}$, this approach is referred as *matrix factorization* (MF) approach in the literature. In this chapter, we consider the approach that attempts to find a low-rank matrix by solving a rank minimization problem, and we will investigate the MF approach in the next chapter.

To effectively find a matrix with minimum rank, in this chapter, we propose a new class of algorithms, called *Iterative weighted Schatten-p norm minimization*

(IS-p), with $1 \leq p \leq 2$, to approximate rank minimization with weighted Schatten-p norm minimization, with $p = 1$ representing the nuclear norm that achieves better approximation to the rank, and $p = 2$ representing the Frobenius norm with more efficient computation. The proposed algorithm turns out to be a generalization of a number of previously proposed iterative re-weighted algorithms [86] based on either only the nuclear norm or only the Frobenius norm to a flexible class of algorithms that can trade optimization accuracy off for computational efficiency, depending on the application requirements. We prove that our algorithms can converge for any p between 1 and 2. Simulations based on synthesized low-rank matrices have shown that our algorithms are more robust than a number of state-of-the-art matrix completion algorithms, including Singular Value Thresholding (SVT) [37], Iterative Reweighted Least Squares Minimization (IRLS-p) [86] and DMFSGD Matrix Completion [43], in both noisy and noiseless scenarios.

We further extend our proposed IS-p algorithm to tensor completion. Specifically, we consider the problem of estimating missing entries of a 3D tensor that has "low-rank" property. Similar to rank minimization in matrix completion, to complete the missing entries in a tensor and especially those in the current timeframe, we minimize a weighted sum of the ranks of three matrices, each unfolded from the tensor along a different dimension. We then extend the proposed IS-p algorithm to solve this approximate tensor completion problem, which again leads to convex optimization that can be efficiently solved.

The results in Chapter 3 have been published in our TON paper [39].

3.2 Robust Matrix Completion via Schatten-p Norm Minimization

In this section, we formulate the robust matrix completion problem, and then introduce our algorithm. Formally, given a noisy input matrix $\mathbf{M} \in \mathbb{R}^{m \times n}$ with missing entries, the problem of low-rank matrix completion is to find a complete matrix $\hat{\mathbf{M}}$ by solving

$$\begin{aligned} \underset{\hat{\mathbf{M}} \in \mathbb{R}^{m \times n}}{\text{minimize}} \quad & \text{rank}(\hat{\mathbf{M}}) \\ \text{subject to} \quad & |\hat{\mathbf{M}}_{i,j} - \mathbf{M}_{i,j}| \leq \tau, \ (i,j) \in \Omega, \end{aligned} \tag{3.1}$$

where τ is a parameter to control the error tolerance on known entries of the input matrix $\mathbf{M}$ [87] or the maximum noise that is present in the observation of each

known pair $(i, j) \in \Omega$. It is well-known that problem (3.1) is an NP-hard problem. In contrast to matrix factorization [43], the advantage of the matrix completion formulation above is that we do need to assume the rank of the network feature matrix is known *a priori*.

One popular approach to solve (3.1) is to use the sum of singular values of $\hat{\mathbf{M}}$, i.e., the nuclear norm, to approximate its rank. The nuclear norm is proved to be the convex envelope of the rank [36] and can be minimized by a number of algorithms, including the well-known singular value thresholding (SVT) [37]. Other smooth approximations include Reweighted Nuclear Norm Minimization [88], and Iterative Reweighted Least Squares algorithm IRLS-p (with $0 \leq p \leq 1$) [86], which attempts to minimize a weighted Frobenius norm of $\hat{\mathbf{M}}$.

3.2.1 A Family of Iterative Weighted Algorithms

Note that all the state-of-the-art rank minimization algorithms mentioned above either minimize the nuclear norm, which is a better approximation to rank, or the Frobenius norm, which is efficient to solve. In this thesis, we propose a family of robust algorithms, called Iterative weighted Schatten-p norm minimization (IS-p), with $1 \leq p \leq 2$, which is a generalization of a number of previous "iterative reweighted" algorithms to a tunable framework; the IS-p algorithm minimizes a reweighted nuclear norm if $p = 1$ and minimizes a reweighted Frobenius norm if $p = 2$. We will show that with IS-p is more robust to any practical parameter settings and trades complexity off for accuracy depending on the application requirements.

The IS-p algorithm is described in Algorithm 1. Note that when $p = 1$, problem (3.2) is a nuclear-norm minimization problem, and when $p = 2$, problem (3.2) becomes Frobenius-norm minimization. In fact, for $1 \leq p \leq 2$, problem (3.2) is a convex problem in general. To see this, for any $\mathbf{M} \in \mathbb{R}^{m \times n}$ $(m \leq n)$. Then, we have $\|\mathbf{M}\|_p^p = \sum_{i=1}^{m} (\sigma_i(\mathbf{M}))^p = \text{tr}\left((\mathbf{M}^\top \mathbf{M})^{\frac{p}{2}}\right)$, which is a convex function for $p \geq 1$, since $\text{tr}(\mathbf{M}^\top \mathbf{M})^{\frac{p}{2}}$ is convex and non-decreasing for $p \geq 1$ [88]. A large number of efficient solutions have been proposed to solve the nuclear-norm and Frobenius-norm versions of (3.2) [37, 88], while for $1 \leq p \leq 2$ problem (3.2) is convex in general. Therefore, we resort to existing methods to solve the convex problem (3.2), which will not be the focus of this thesis. Furthermore, exact singular value decomposition for $\hat{\mathbf{M}}^k$ in Step 6 can be performed within polynomial time with a complexity of $O(m^2 n)$.

Algorithm 1 The IS-p Algorithm ($1 \le p \le 2$)

1: **Input**: An incomplete matrix $\mathbf{M} \in \mathbb{R}^{m \times n}$ ($m \le n$) with entries known only for $(i,j) \in \Omega$; the error tolerance τ on known entries
2: **Output**: $\hat{\mathbf{M}}$ as an approximate solution to (3.1).
3: Initially, $\mathbf{L}^0 := \mathbf{I}$, δ^0 is an arbitrary positive number
4: **for** $k = 1$ to maxIter **do**
5: Solve the following convex optimization problem to obtain the optimal solution $\hat{\mathbf{M}}^k$:

$$\begin{aligned} \underset{\hat{\mathbf{M}}}{\text{minimize}} \quad & \|\mathbf{L}^{k-1}\hat{\mathbf{M}}\|_p^p \\ \text{subject to} \quad & |\hat{\mathbf{M}}_{i,j} - \mathbf{M}_{i,j}| \le \tau, \ (i,j) \in \Omega \end{aligned} \tag{3.2}$$

6: $[\mathbf{U}^k, \mathbf{\Sigma}^k, \mathbf{V}^k] := \text{SVD}(\hat{\mathbf{M}}^k)$, where $\mathbf{\Sigma}^k$ is an $m \times n$ diagonal matrix with non-negative real numbers (singular values of $\hat{\mathbf{M}}^k$) $\sigma_1^k, \ldots, \sigma_m^k$ on the diagonal
7: Form a weight matrix $\mathbf{W}^k \in \mathbb{R}^{m \times m}$, where

$$\mathbf{W}_{ij}^k := \begin{cases} \left((\sigma_i^k)^p + \delta^{k-1}\right)^{-\frac{1}{p}}, & i = j \\ 0, & i \ne j \end{cases}$$

8: Choose δ^k such that $0 < \delta^k \le \delta^{k-1}$.
9: $\mathbf{L}^k := \mathbf{U}^k \mathbf{W}^k {\mathbf{U}^k}^\top$
10: **end for**
11: $\hat{\mathbf{M}} := \hat{\mathbf{M}}^{\text{maxIter}}$

Let us now provide some mathematical intuition to explain why Algorithm 1 can approximate the rank minimization. Initially, we replace the objective function $\text{rank}(\hat{\mathbf{M}})$ with $\|\hat{\mathbf{M}}\|_p$. Subsequently, in each iteration k, we are minimizing $\|\mathbf{L}^{k-1}\hat{\mathbf{M}}\|_p^p$. Recall that in Step 6 of iteration k, the optimal solution $\hat{\mathbf{M}}^k$ can be factorized as $\hat{\mathbf{M}}^k = \mathbf{U}^k\mathbf{\Sigma}^k\mathbf{V}^k$ via singular value decomposition, where $\mathbf{U}^k \in \mathbb{R}^{m \times m}$ and $\mathbf{V}^k \in \mathbb{R}^{n \times n}$ are unitary square matrices, i.e., ${\mathbf{U}^k}^\top\mathbf{U}^k = \mathbf{I}$, ${\mathbf{V}^k}^\top\mathbf{V}^k = \mathbf{I}$. Thus, we have $\|\mathbf{L}^{k-1}\hat{\mathbf{M}}^k\|_p^p = \|\mathbf{U}^{k-1}\mathbf{W}^{k-1}{\mathbf{U}^{k-1}}^\top\mathbf{U}^k\mathbf{\Sigma}^k\mathbf{V}^k\|_p^p$. If $\mathbf{U}^{k-1} \approx \mathbf{U}^k$ after a number of iterations, we will have

$$\begin{aligned} \|\mathbf{L}^{k-1}\hat{\mathbf{M}}^k\|_p^p &\approx \|\mathbf{U}^{k-1}\mathbf{W}^{k-1}{\mathbf{U}^k}^\top\mathbf{U}^k\mathbf{\Sigma}^k\mathbf{V}^k\|_p^p \\ &= \|U^{k-1}(W^{k-1}\Sigma^k)V^k\|_p^p \\ &= \sum_{i=1}^m \left(\sigma_i\left(\mathbf{W}^{k-1}\mathbf{\Sigma}^k\right)\right)^p \\ &= \sum_{i=1}^m \left(\frac{\sigma_i^k}{((\sigma_i^{k-1})^p + \delta^{k-1})^{1/p}}\right)^p \\ &= \sum_{i=1}^m \frac{(\sigma_i^k)^p}{(\sigma_i^{k-1})^p + \delta^{k-1}}, \end{aligned} \tag{3.3}$$

which eventually approaches $\text{rank}(\hat{\mathbf{M}}^k)$. To see this, note that for two sufficiently

small positive constants δ^{k-1} and δ^k, upon convergence, i.e., when $\sigma_i^k = \sigma_i^{k-1}$, we have

$$\frac{(\sigma_i^k)^p}{(\sigma_i^{k-1})^p + \delta^{k-1}} \approx \frac{(\sigma_i^k)^p}{(\sigma_i^k)^p + \delta^k} \approx \begin{cases} 0 & \text{if } \sigma_i^k = 0, \\ 1 & \text{if } \sigma_i^k > 0, \end{cases}$$

Therefore, $\|\mathbf{L}^{k-1}\hat{\mathbf{M}}^k\|_p^p$ represents the number of nonzero singular values σ_i^k in $\hat{\mathbf{M}}^k$, which is exactly the rank of $\hat{\mathbf{M}}^k$.

3.3 Convergence Analysis

The above informal analysis only provides an intuitive explanation as to why the algorithm works, based on the hope that the algorithm will converge. The following theorem can ensure the convergence of the produced rank($\hat{\mathbf{M}}_k$) and therefore guarantee the convergence of Algorithm 1.

Theorem 3.3.1. *Suppose $\hat{\mathbf{M}}^k$ is the output of Algorithm 1 in iteration k. For any matrix $\mathbf{M} \in \mathbb{R}^{m\times n}$ and any $p \in [1, 2]$, rank($\hat{\mathbf{M}}^k$) converges. In particular, for a sufficiently large k, we have $\sigma_i(\hat{\mathbf{M}}^k) - \sigma_i(\hat{\mathbf{M}}^{k-1}) \to 0$, for $i = 1, \ldots, m$.*

Proof. We first present some useful lemmas.

Lemma 3.3.1. *For any $\mathbf{A} \in \mathbb{R}^{m\times n}$ and $\mathbf{B} \in \mathbb{R}^{n\times r}$, the following holds for all $1 \le p \le 2$:*

$$\sum_{i=1}^{n} \sigma_{n-i+1}^p(\mathbf{A})\sigma_i^p(\mathbf{B}) \le \|\mathbf{AB}\|_p^p \le \sum_{i=1}^{n} \sigma_i^p(\mathbf{A})\sigma_i^p(\mathbf{B}), \tag{3.4}$$

where $\sigma_i(\mathbf{A})$ denotes the ith singular value of $\mathbf{A}$.

We leave the proof of this lemma in Sec. 3.7.

Corollary 3.3.1. *Given an $m \times m$ diagonal matrix $\mathbf{A}$ with nonnegative and non-decreasing (non-increasing) diagonal entries $a_{11}, \ldots, a_{mm}$, and another $m \times n$ diagonal matrix $\mathbf{B}$ with nonnegative and non-increasing (non-decreasing) diagonal entries $b_{11}, \ldots, b_{mm}$, we have $\|\mathbf{AUB}\|_p \ge \|\mathbf{AB}\|_p$ for any $m \times m$ square unitary matrix $\mathbf{U}$ (i.e., $\mathbf{U}^\top = \mathbf{I}$), where $1 \le p \le 2$.*

Proof of Corollary 3.3.1. Without loss of generality, we assume $a_{11} \ge a_{22} \ge \ldots \ge a_{mm} \ge 0$ and $0 \le b_{11} \le b_{22} \le \ldots \le b_{mm}$. By Lemma 3.3.1, we have

$$\begin{aligned} \|\mathbf{AUB}\|_p^p &\ge \sum_{i=1}^{m} \sigma_i^p(\mathbf{A})\sigma_{n-i+1}^p(\mathbf{UBI}) = \sum_{i=1}^{m} \sigma_i^p(\mathbf{A})\sigma_{n-i+1}^p(\mathbf{B}) \\ &= \sum_{i=1}^{m} a_{ii}^p b_{ii}^p = \sum_{i=1}^{m} \sigma_i^p(\mathbf{AB}) = \|\mathbf{AB}\|_p^p, \end{aligned}$$

proving the corollary. □

We now prove Theorem 3.3.1. According to Corollary 3.3.1 and the *unitarily invariant property* of Schatten-p norms, we have

$$\begin{aligned}
\|\mathbf{L}^{k-1}\hat{\mathbf{M}}^k\|_p &= \|\mathbf{U}^{k-1}\mathbf{W}^{k-1}{\mathbf{U}^{k-1}}^\top\mathbf{U}^k\mathbf{\Sigma}^k{\mathbf{V}^k}^\top\|_p && (3.5)\\
&= \|\mathbf{W}^{k-1}{\mathbf{U}^{k-1}}^\top\mathbf{U}^k\mathbf{\Sigma}^k\|_p && (3.6)\\
&\geq \|\mathbf{W}^{k-1}\mathbf{\Sigma}^k\|_p && (3.7)\\
&= \left(\sum_{i=1}^{n}\frac{\left(\sigma_i^k\right)^p}{\left(\sigma_i^{k-1}\right)^p+\delta^{k-1}}\right)^{\frac{1}{p}}, && (3.8)
\end{aligned}$$

where (3.7) is due to Lemma 3.3.1, since $\mathbf{W}^{k-1}$ and $\mathbf{\Sigma}^k$ are diagonal matrices with nonnegative non-decreasing and non-increasing entries, respectively, and ${\mathbf{U}^{k-1}}^\top\mathbf{U}^k$ is still unitary.

Since $\hat{\mathbf{M}}^k$ is the optimal solution to (3.2), we have

$$\begin{aligned}
\|\mathbf{L}^{k-1}\hat{\mathbf{M}}^k\|_p &\leq \|\mathbf{L}^{k-1}\hat{\mathbf{M}}^{k-1}\|_p && (3.9)\\
&= \|\mathbf{U}^{k-1}\mathbf{W}^{k-1}\mathbf{\Sigma}^{k-1}{\mathbf{V}^{k-1}}^\top\|_p && (3.10)\\
&= \|\mathbf{W}^{k-1}\mathbf{\Sigma}^{k-1}\|_p && (3.11)\\
&= \left(\sum_{i=1}^{n}\frac{\left(\sigma_i^{k-1}\right)^p}{\left(\sigma_i^{k-1}\right)^p+\delta^{k-1}}\right)^{\frac{1}{p}} && (3.12)
\end{aligned}$$

Since $\delta^k \leq \delta^{k-1}$, we have

$$\begin{aligned}
\sum_{i=1}^{n}\frac{\left(\sigma_i^k\right)^p}{\left(\sigma_i^{k-1}\right)^p+\delta^{k-1}} &\leq \sum_{i=1}^{n}\frac{\left(\sigma_i^{k-1}\right)^p}{\left(\sigma_i^{k-1}\right)^p+\delta^{k-1}},\\
\sum_{i=1}^{n}\frac{\left(\sigma_i^k\right)^p+\delta^k}{\left(\sigma_i^{k-1}\right)^p+\delta^{k-1}} &\leq \sum_{i=1}^{n}\frac{\left(\sigma_i^{k-1}\right)^p+\delta^{k-1}}{\left(\sigma_i^{k-1}\right)^p+\delta^{k-1}} = n.
\end{aligned}$$

Let $x_i^k := (\sigma_i^k)^p$ and $\mathbf{x}^k = (x_1^k, x_2^k, ..., x_n^k)$. Define a function $\mathcal{L}: \mathbb{R}^n \to \mathbb{R}_+$, $\mathcal{L}(\mathbf{x}) = \prod_{i=1}^n (x_i + \delta^k)$, with $\delta^k > 0$. We will show that the sequence $\mathcal{L}(\mathbf{x}^k)$ is monotonically non-increasing using a similar method in [89], and prove the convergence of σ_i^k for $1 \leq i \leq n$.

Using the inequality between the arithmetic and geometric means for nonnegative terms, we have

$$\prod_{i=1}^{n}\frac{x_i^k+\delta^k}{x_i^{k-1}+\delta^{k-1}} \leq 1, \tag{3.13}$$

which implies that $\mathcal{L}(\mathbf{x}^k) \leq \mathcal{L}(\mathbf{x}^{k-1})$. Also, since $x_i^k \geq 0$, $\mathcal{L}$ is bounded below by δ^n, the sequence $\mathcal{L}(\mathbf{x}^k)$ converges. It implies that

$$\prod_{i=1}^{n} \frac{x_i^k + \delta^k}{x_i^{k-1} + \delta^{k-1}} = \frac{\mathcal{L}(\mathbf{x}^k)}{\mathcal{L}(\mathbf{x}^{k-1})} \to 1. \tag{3.14}$$

Define $\mathbf{y}^k$ to be

$$y_i^k = \frac{x_i^k + \delta^k}{x_i^{k-1} + \delta^{k-1}},$$

and $y_1^k = 1 + \epsilon$. We have

$$\prod_{i=1}^{n} y_i^k = (1+\epsilon) \prod_{i=2}^{n} y_i^k \leq (1+\epsilon) \left(1 - \frac{\epsilon}{n-1}\right)^{n-1} = f(\epsilon) \tag{3.15}$$

by combining $\sum_{i=1}^{n} y_i^k \leq n$ and the inequality between the arithmetic and geometric means. Function $f(\epsilon)$ is continuous and satisfies $f(0) = 1$, $f'(0) = 0$, and $f''(\epsilon) < 0$, for $|\epsilon| < 1$. Hence, $f(\epsilon) < 1$ for $\epsilon \neq 0$, $|\epsilon| < 1$.

Therefore, since $\prod_{i=1}^{n} y_i^k \to 1$, we have $f(\epsilon) \to 1$, which in turn implies $\epsilon \to 0$. Hence $y_1^k \to 1$, and the same holds for all y_i^k. Thus, we have

$$y_i^k = \frac{(\sigma_i^k)^p + \delta^k}{(\sigma_i^{k-1})^p + \delta^{k-1}} \to 1.$$

By monotone convergence theorem, there exists a point $\delta^* \geq 0$ such that $\delta^k \to \delta^*$, and thus $\delta^{k-1} - \delta^k \leq \delta^{k-1} - \delta^* \to 0$, implying $\delta^{k-1} - \delta^k \to 0$. Since σ_i^k is finite, we conclude that $\sigma_i^k - \sigma_i^{k-1} \to 0$, for all $i = 1, \ldots, n$, which implies $\text{rank}(\hat{\mathbf{M}}^k) - \text{rank}(\hat{\mathbf{M}}^{k-1}) \to 0$. □

3.3.1 Relationships to Prior Algorithms

We now point out that the proposed IS-p algorithm is a generalization of a number of previous reweighted approximate algorithms based on either nuclear norm or Frobenius norm alone to a tunable class of algorithms trading complexity off for performance.

Singular value thresholding (SVT) is an algorithm to solve the convex nuclear norm minimization:

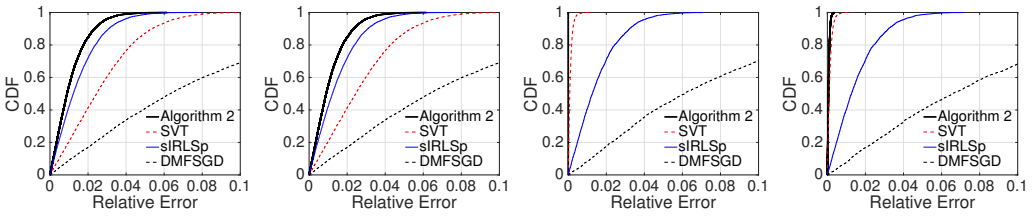
$$\begin{aligned} \underset{\hat{\mathbf{M}} \in \mathbb{R}^{m \times n}}{\text{minimize}} \quad & \|\hat{\mathbf{M}}\|_* \\ \text{subject to} \quad & |\hat{\mathbf{M}}_{i,j} - \mathbf{M}_{i,j}| \leq \tau, \ (i,j) \in \Omega, \end{aligned} \tag{3.16}$$

which approximates (3.1). It is shown [90] that for most matrices of rank r, (3.16) yields the same solution as (3.1), provided that the number of known entries $m \geq$

$Cn^{6/5}r\log n$ for some positive constant C. However, when $m < Cn^{6/5}r\log n$, the nuclear-norm-minimizing solution from SVT usually cannot approximate (3.1) well. In fact, SVT can be viewed as only performing the first iteration of the proposed Algorithm 1 with $p = 1$. In contrast, Algorithm 1 adopts multiple iterations of reweighted minimizations to refine the results and can further approximate the rank minimization problem over iterations, even if $m < Cn^{6/5}r\log n$.

A number of iterative reweighted approximations to (3.1) have been proposed. They could be different in performance, mainly due to the different norms (either Frobenius norm or nuclear norm) adopted as well as the way to form the weight matrix L^k. Iterative Reweighted Least Squares (IRLS-p and sIRLS-p) [88] is also a reweighted algorithm to approximate the affine rank minimization problem (i.e., problem (3.1) with $\tau = 0$ in the constraint). It minimizes a weighted Frobenius norm $\|\mathbf{L}^{k-1}\mathbf{M}\|_F$ in each iteration k to produce an $\mathbf{M}^k$, where $\mathbf{L}^{k-1} := \sqrt{(\mathbf{M}^{k-1}{}^\top\mathbf{M}^{k-1} + \delta\mathbf{I})^{p/2-1}}$ with $0 \le p \le 1$. By simple maths derivations, we find the weight $\mathbf{L}^{k-1}$ in IRLS-p is different from that in Algorithm 1, therefore yielding different approximation results. Furthermore, IRLS-p can only minimize a Frobenius norm in each iteration, whereas the nuclear norm is known to be the best convex approximation of the rank function [36]. In contrast, the proposed Algorithm 1 represents a family of algorithms including nuclear norm minimization (when $p = 1$) on one end to achieve better approximation and Frobenius norm minimization (when $p = 2$) on the other end for faster computation.

Furthermore, [54] presents another iterative reweighted algorithm that essentially minimizes $\|\mathbf{L}_1^k\mathbf{M}\mathbf{L}_2^k\|_*$, which can only minimize the nuclear norm but not Frobenius norms, and thus takes longer time to compute in each iteration. [91] presents a reweighted nuclear norm minimization scheme which essentially morphs to minimizing $\|\mathbf{L}^{k-1}\mathbf{M}\|_*$ in each iteration, where $\mathbf{L}^k = \mathbf{U}^k(\mathbf{\Sigma}^k + \delta\mathbf{I})^{-1}\mathbf{U}^k$, where as in our algorithm, $\mathbf{L}^k = \mathbf{U}^k((\mathbf{\Sigma}^k)^p + \delta\mathbf{I})^{-1/p}\mathbf{U}^k{}^\top$. When $p = 1$, note that the only difference here is that a $\mathbf{U}^k{}^\top = \mathbf{U}^{k-1}$ is multiplied at the end, whereas in [91] U^k itself is multiplied. In fact, Theorem 3.3.1 holds for any $\mathbf{L}^k = \mathbf{U}^k((\mathbf{\Sigma}^k)^p + \delta\mathbf{I})^{-1/p}\mathbf{Q}$, where $\mathbf{Q}$ is unitary (i.e., $\mathbf{Q}^\top\mathbf{Q} = \mathbf{I}$), since (3.7) will hold for any unitary $\mathbf{Q}$, including $\mathbf{Q} = \mathbf{U}^k$ and $\mathbf{Q} = \mathbf{U}^k{}^\top$, according to Corollary 3.3.1 in the appendix. However, when $\mathbf{Q} = \mathbf{U}^k{}^\top$, as has been shown in (3.3), $\|\mathbf{L}^{k-1}\hat{\mathbf{M}}^k\|$ will eventually approach $\text{rank}(\hat{\mathbf{M}}^k)$, yielding better approximation.



(a) $R = 0.3$, ϵ=0 (b) $R = 0.3$, ϵ=0.1 (c) $R = 0.7$, ϵ=0 (d) $R = 0.7$, ϵ=0.1

Figure 3.1: Performance of IS-p ($p = 1$) and other algorithms on synthetic 100×100 matrices with rank $r = 20$, under sample rates $R = 0.3$ and $R = 0.7$.

3.4 Performance on Synthesized Low-Rank Data

We evaluate our algorithm based on synthesized true low-rank matrices contaminated by random noises, in comparison with several state-of-the-art approaches to matrix completion:

- **Singular Value Thresholding (SVT)** [37]: an algorithm for nuclear norm minimization as an approximation to rank minimization;
- **Iterative Reweighted Least Squares (sIRLS-p)** [88]: an iterative algorithm to approximate rank minimization with a reweighted Frobenius-norm minimization in each iteration. According to [88], the performance of sIRLS-1 is proved to guarantee the recovery performance, thus we choose sIRLS-1 for comparison;
- **DMFSGD Matrix Factorization** [43]: a distributed network latency prediction algorithm that attempts to approximate a given matrix $\mathbf{M}$ using the product of two smaller matrices $\hat{\mathbf{M}} = \mathbf{U}\mathbf{V}^\top$, where $\mathbf{U} \in \mathbb{R}^{n \times r}$ and $\mathbf{V} \in \mathbb{R}^{n \times r}$, such that a loss function based on $\mathbf{M} - \hat{\mathbf{M}}$ is minimized, where r is the assumed rank of $\hat{\mathbf{M}}$.

In our experiments, we randomly generate 100×100 matrices with rank $r = 20$, contaminated by noise. The generated matrix can be represented as $\mathbf{X} = \mathbf{U}\mathbf{V}^\top + \epsilon\mathbf{N}$, where $\mathbf{U}$ and $\mathbf{V}$ are randomly generated $n \times r$ matrices ($n = 100$, $r = 20$) with entries uniformly distributed between 0 and 1. $\mathbf{N}$ is an $n \times n$ standard Gaussian noise matrix. We run simulations under the sample rates $R = 0.3$ and $R = 0.7$ and under both the noiseless case $\epsilon = 0$ and the noisy case $\epsilon = 0.1$ to test the algorithm robustness.

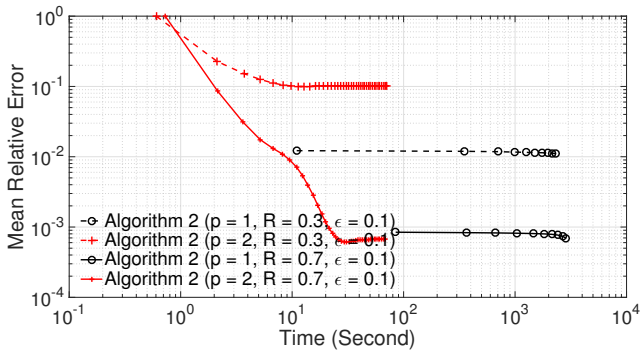


Figure 3.2: A comparison between IS-1 (the nuclear-norm version) and IS-2 (the Frobenius-norm version) in terms of recovery errors and running time.

Fig. 3.1(a) and Fig. 3.1(c) compare the performance of different algorithms in the noiseless case. As we can see, our algorithm is the best at low sample rate ($R = 0.3$). When the sample rate is high ($R = 0.7$), both our algorithm and SVT are the best. For the noisy case, Fig. 3.1(b) and Fig. 3.1(d) show that our algorithm outperforms all other algorithms at both the low sample rate ($R = 0.3$) and the high sample rate ($R = 0.7$), thus proving that our algorithm is the most robust to noise.

Under the same setting, we now investigate the tradeoff between setting $p = 1$ (the Nuclear norm version) and $p = 2$ (the Frobenius norm version) in IS-p in Fig. 3.2. In general, the nuclear norm version (IS-1) usually converges in a few iterations (usually one iteration) and more iterations will give little improvement. On the other hand, the Frobenius norm version (IS-2) requires more iterations to converge, and the relative recovery error decreases significantly as more iterations are adopted.

Specifically, under $R = 0.3$, IS-1 already achieves a low error within about 10 seconds. In this case, although IS-2 leads to a higher error, yet it enables a tunable tradeoff between accuracy and running time. When $R = 0.7$, IS-2 is better considering both the running time and accuracy. Therefore, we make the following conclusions:

First, IS-1 (the nuclear norm version) achieves better accuracy in general, yet at the cost of a higher complexity. IS-1 could be slower when more training data is available. The reason is that when problem (3.2) for $p = 1$ in Algorithm 1 is solved by a semidefinite program (SDP) (with performance guarantees [36]), which could be slow when data size increases. Note that SVT or other first-order algorithms

cannot be applied to (3.2) due to the weight matrix $\mathbf{L}$ in the objective. Therefore, IS-1 should only be used upon abundant computational power or high requirement on accuracy.

Second, IS-2 has a low per-iteration cost, i.e., the error decreases gradually when more iterations are used. Therefore, it allows the system operator to flexibly tune the achieved accuracy by controlling the running time invested. Furthermore, although IS-2 does not always lead to the best performance, the achieved relative error is usually sufficient. Due to this flexible nature of IS-2, we set $p = 2$ for our experiments on network latency estimation in Sec. 3.6, so that we can control the rank of the recovered matrix that we want to achieve, under a given budget of running time.

In our experiments, we actually set $\delta^k = \delta^{k-1}/\eta$, where $\eta > 1$ is a constant. We find that good performance is usually achieved by a large initial value of δ and an appropriate η. Specifically, we set the initial δ to 100,000 and $\eta = 2$.

3.5 Extension to Tensor Approximation

In this section, we extend our proposed IS-p algorithm to tensor completion. In particular, we use a *tensor* $\mathcal{M} = (\mathcal{M}_{i,j,t}) \in \mathbb{R}^{m\times n\times T}$ to represent a 3-dimensional array that consists of T matrices with missing values. Examples of 3D tensors include color images, which consists 3 channels that represent red, green and blue, and network round-trip time (RTT) matrices in different time periods. For simplicity, we call each $m \times n$ matrix as a "frame" or "channel" in this section. Note that we focus on 3D tensors for brevity and it is natural to extend to higher order tensors. Let $\mathbf{\Omega}$ denote the set of indices (i, j, t) where the measurements $\mathcal{M}_{i,j,t}$ are known and $\mathbf{\Theta}$ denote the set of unknown indices. The problem is to recover missing values in $\mathcal{M}$.

In order to complete all missing values in $\mathcal{M}$, we generalize the matrix completion problem to tensor completion and extend our IS-p algorithm to the tensor case. Here we only focus on tensors in $\mathbb{R}^{n\times n\times T}$, a size relevant to our specific latency estimation problem, although our idea can be applied to general tensors. Given a tensor $\mathcal{M} \in \mathbb{R}^{n\times n\times T}$ with missing entries, tensor completion aims to find a complete low-rank tensor $\hat{\mathcal{M}}$ by solving

$$\begin{array}{ll} \underset{\hat{\mathcal{M}}\in\mathbb{R}^{n\times n\times T}}{\text{minimize}} & \text{rank}(\hat{\mathcal{M}}) \\ \text{subject to} & |\hat{\mathcal{M}}_{i,j,t} - \mathcal{M}_{i,j,t}| \leq \tau,\ (i,\ j,\ t) \in \mathbf{\Omega}, \end{array} \tag{3.17}$$

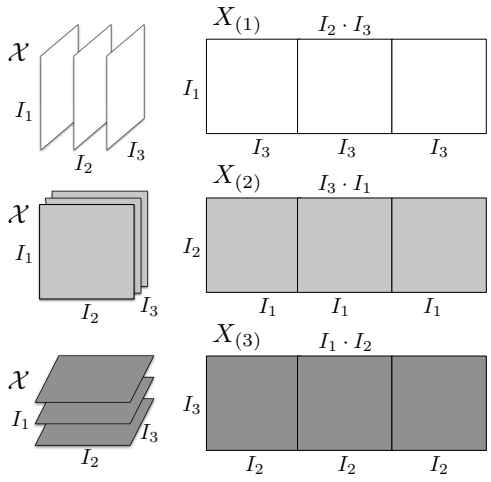


Figure 3.3: Illustration of tensor unfolding for the 3D case.

where τ is a parameter to control the error tolerance on known entries. However, unlike the case of matrices, the problem of finding a low rank approximation to a tensor is ill-posed. More specifically, it has been shown that the space of rank-r tensors is non-compact [92] and that the nonexistence of low-rank approximations occurs for many different ranks and orders. In fact, even computing the rank of a general tensor (with a dimension≥ 3) is an NP hard problem [34] and there is no known explicit expression for the convex envelope of the tensor rank.

A natural alternative is to minimize a weighted sum of the ranks of some 2D matrices "unfolded" from the 3D tensor, hence reducing tensor completion to matrix completion. The unfold operation is illustrated in Fig. 3.3 for a 3D tensor along each of the three dimensions. Here I_1, I_2 and I_3 are index sets for each dimension. These unfolded matrices can be computed as follows:

- The column vectors of $\mathcal{M}$ are column vectors of $\mathbf{M}_{(1)} \in \mathbb{R}^{m \times nT}$.
- The row vectors of $\mathcal{M}$ are column vectors of $\mathbf{M}_{(2)} \in \mathbb{R}^{n \times mT}$.
- The (depth) vectors on the third dimension of $\mathcal{M}$ are column vectors of $\mathbf{M}_{(3)} \in \mathbb{R}^{T \times mn}$.

With unfolding operations defined above, the problem of "low-rank" tensor approximation can be formulated as minimizing the weighted sum of ranks for all three

unfolded matrices [93]:

$$\begin{aligned} \underset{\hat{\mathcal{M}}\in\mathbb{R}^{n\times n\times T}}{\text{minimize}} \quad & \sum_{l=1}^{3} \alpha_l \cdot \text{rank}(\hat{\mathbf{M}}_{(l)}) \\ \text{subject to} \quad & |\hat{\mathcal{M}}_{i,j,t} - \mathcal{M}_{i,j,t}| \leq \tau,\ (i,\ j,\ t) \in \mathbf{\Omega}, \end{aligned} \tag{3.18}$$

where α_l is a convex combination coefficient, with $\alpha_l \geq 0$ and $\sum_{l=1}^{3} \alpha_l = 1$.

Apparently, the above nonconvex problem of minimizing the weighted sum of ranks is still hard to solve. We propose a generalization of the proposed IS-p algorithm to the tensor case. Our "low-rank" tensor approximation algorithm is described in Algorithm 2. The algorithm first solves a convex optimization problem by minimizing the sum of weighted Schatten-p norms of all unfolded matrices within the given noise tolerance. Here the weight matrices $\mathbf{L}_{(l)}$ are assigned for each unfolded matrix of tensor $\mathcal{X}$. Then the algorithm will update weight matrices $\mathbf{L}_{(l)}$ one by one. This procedure is similar to what we did in 2D matrix completion.

Algorithm 2 IS-p Algorithm for Tensor Completion

1: Initialize $\mathbf{L}_{(l)}^0 := \mathbf{I},\ p,\ \delta_{(l)}^0,\ \tau_{(l)},\ \eta_{(l)},\ l = 1, 2, 3$
2: **for** $k = 1$ to maxIter **do**
3: Solve the following convex optimization problem to obtain the optimal solution $\hat{\mathcal{M}}^k$:

$$\begin{aligned} \underset{\hat{\mathcal{M}}}{\text{minimize}} \quad & \sum_{l=1}^{3} \alpha_l \|\mathbf{L}_{(l)}^{k-1} \hat{\mathbf{M}}_{(l)}\|_p^p \\ \text{subject to} \quad & |\hat{\mathcal{M}}_{ijt} - \mathcal{M}_{ijt}| \leq \tau,\ (i,j,t) \in \mathbf{\Omega} \end{aligned} \tag{3.19}$$

4: **for** $l = 1$ to 3 **do**
5: $[\mathbf{U}_{(l)}^k, \mathbf{\Sigma}_{(l)}^k, \mathbf{V}_{(l)}^k] := SVD\left(\hat{\mathbf{M}}_{(l)}^k\right)$, where $\mathbf{\Sigma}_{(l)}^k$ is a diagonal matrix with diagonal elements of $\{\sigma_{(l),i}^k\}$.
6: $\mathbf{W}_{(l),ij}^k := \begin{cases} \left(\left(\sigma_{(l),i}^k\right)^p + \delta_{(l)}^{k-1}\right)^{-\frac{1}{p}}, & i = j \\ 0, & i \neq j \end{cases}$
7: $\mathbf{L}_{(l)}^k := \mathbf{U}_{(l)}^k \mathbf{W}_{(l)}^k \mathbf{U}_{(l)}^{k\ \top}$
8: Choose $\delta_{(l)}^k$ such that $0 < \delta_{(l)}^k \leq \delta_{(l)}^{k-1}$.
9: **end for**
10: **end for**
11: $\hat{\mathbf{M}} := \hat{\mathbf{M}}^{\text{maxIter}}$

It is not hard to check that problem (3.19) is a convex problem for all $1 \leq p \leq 2$, since for a fixed weight matrix $\mathbf{L}$, $||\mathbf{LX}||_p^p$ is a convex function of $\mathbf{X}$. In (3.19), we can see that the objective function is a convex combination of three convex functions. Note that the convergence of Algorithm 2 cannot be extended directly

from the matrix case, but we observe in simulation that our algorithm has robust convergence performance.

3.6 Performance Evaluation

We evaluate our proposed network latency estimation approaches on both single frames of $2D$ RTT matrices and 3D multi-frame RTT measurements, in comparison with a number of state-of-the-art latency estimation algorithms. For network latency prediction based on 2D data, we evaluate our algorithm on the Seattle dataset and PlanetLab dataset; for dynamic network latency prediction based on 3D data, our algorithm is evaluated based on the Seattle dataset[1].

3.6.1 Single-Frame Matrix Completion

We define the relative estimation error (RE) on missing entries as $|\hat{\mathbf{M}}_{ij} - \mathbf{M}_{ij}|/\mathbf{M}_{ij}$, for $(i, j) \in \Omega$, which will be used to evaluate prediction accuracy. We compare our algorithm with the following approaches:

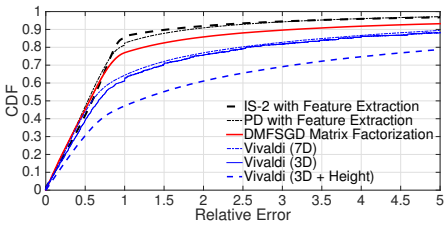
- **Vivaldi** with dimension $d = 3$, $d = 7$, and $d = 3$ plus a height parameter;
- **DMFSGD Matrix Factorization** is a matrix factorization approach for RTT prediction under an assumed rank, and
- **PD with feature extraction** as shown in [94], which uses Penalty Decomposition for matrix completion with feature extraction [39].

When estimating network latencies, [39] have shown that it is beneficial to do low-rank matrix completion on *feature matrix* instead of directly on *latency matrix*. In particular, the network latency matrix $\mathbf{M}$ is regarded as a composite of two matrices, the distance matrix $\mathbf{D}$ and a *low-rank feature matrix* $\mathbf{F}$, by Hadamard product $\mathbf{M} = \mathbf{D} \circ \mathbf{F}$. The feature matrix $\mathbf{F}$ is extracted by running an alternating algorithm and we refer readers to [39] for more details of this approach.

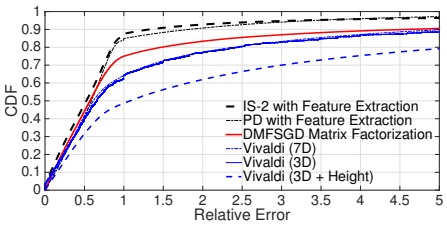
Note that our proposed IS-p approach attempts to recover $\mathbf{F}$, not directly on $\mathbf{M}$. For our method, the Euclidean embedding part in feature extraction is done using Vivaldi with a low dimension of $d = 3$ without the height.

We randomly choose 50 frames from the 688 frames in the Seattle data. For PlanetLab data, as differences among the 18 frames are small, we randomly choose

[1]We have made both datasets publicly available for reproducibility: `https://github.com/uofa-rzhu3/NetLatency-Data`.

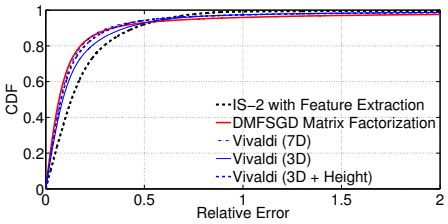


(a) Seattle (Sample rate $R = 0.3$)

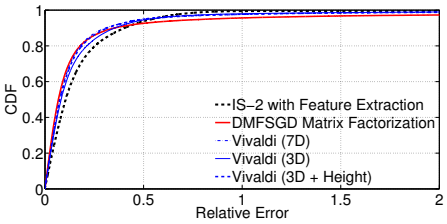


(b) Seattle (Sample rate $R = 0.7$)

Figure 3.4: The CDFs of relative estimation errors on missing values for the Seattle dataset, under sample rates $R = 0.3$ and $R = 0.7$, respectively.



(a) PlanetLab (Sample rate $R = 0.3$)



(b) PlanetLab (Sample rate $R = 0.7$)

Figure 3.5: The CDFs of relative estimation errors on missing values for the PlanetLab dataset, under sample rates $R = 0.3$ and $R = 0.7$, respectively.

one frame to test the methods. Recall that the sample rate R is defined as the percentage of known entries. Each chosen frame is independently sampled at a low rate $R = 0.3$ (70% latencies are missing) and at a high rate $R = 0.7$, respectively.

For DMFSGD, we set the rank of the estimation matrix $\hat{\mathbf{M}}$ to $r = 20$ for Seattle data and $r = 10$ for PlanetLab data, respectively, since the 20^{th} (or 10^{th}) singular value of $\mathbf{M}$ is less than 5% of the largest singular value in Seattle (or PlanetLab). In fact, $r = 10$ is adopted by the original DMFSGD work [43] based on PlanetLab data. We have tried other ranks between 10-30 and observed similar performance. We plot the relative estimation errors on missing latencies in Fig. 3.4 for the Seattle data and Fig. 3.5 for the PlanetLab data. They are both under 5 methods.

For the Seattle results in Fig. 3.4(a) and Fig. 3.4(b), we can see that the IS-2 algorithm with feature extraction outperform all other methods by a substantial margin. We first check the Vivaldi algorithms. Even if Vivaldi Euclidean embedding is performed in a 7D space, it only improves over 3D space slightly, due to the fundamental limitation of Euclidean assumption. Furthermore, the 3D Vivaldi with a height parameter, which models the "last-mile latency" to the Internet core [95], is even worse than the 3D Vivaldi without heights in Seattle. This implies that la-

tencies between personal devices are better modeled by their pairwise core distances multiplied by the network conditions, rather than by pairwise core distances plus a "last-mile latency".

The DMFSGD algorithm is also inferior to our algorithm both because it solely relies on the low-rank assumption, which may not be enough to model the Seattle latency matrices accurately, and because the proposed IS-p algorithm has better performance than DMFSGD in terms matrix completion.

Fig. 3.4 also shows that the proposed IS-2 with feature extraction is even better than our work [94] that adopts the Penalty Decomposition (PD) heuristic for matrix completion after feature extraction, the latter showing the second best performance among all methods on Seattle data. This justifies our adoption of IS-2 as a high-performance algorithm for the matrix completion part, especially for highly unpredictable Seattle latencies.

In contrast, for the PlanetLab results shown in Fig. 3.5(a) and Fig. 3.5(b), our algorithm does not have a clear benefit over other state-of-the-art algorithms. The reason behind is that, the latencies in PlanetLab are symmetric and only a small portion of them violate the triangle inequality. Thus, network coordinate systems such as Vivaldi already have excellent performance. Furthermore, the RTT matrix $\mathbf{M}$ and the distance matrix $\hat{\mathbf{D}}$ have similar singular values. Hence, there is no need to extract the network feature matrix $\mathbf{F}$ for PlanetLab. In this case, performing a distance-feature decomposition could introduce additional errors and is not necessary. These observations again show the unique advantage of our approach to personal device networks, although it could be an overkill for stable PlanetLab nodes.

3.6.2 Multi-Frame Tensor Approximation

We test our multi-frame latency tensor completion approach on 50 groups of consecutive frames in Seattle. Each group contains $T = 3$ consecutive frames of incomplete RTT measurements, forming an incomplete tensor, and such triple-frame groups are randomly selected from the Seattle dataset. The objective is to recover all the missing values in each selected tensor.

Recall that tensor completion is applied on the network feature tensor $\mathcal{F}$, whose unfolding matrices are $\mathbf{F}_{(l)}$ for $l = 1, 2, 3$. Since our tensor has a size of $\mathbb{R}^{n\times n\times T}$, the first two unfolded matrices $\mathbf{F}_{(1)}$ and $\mathbf{F}_{(2)}$ have the same size $n \times nT$. Since $T = 3$

in our experiment, the size of the other unfolded matrix $\mathbf{F}_{(3)}$ is $3 \times n^2$. As the convex combination coefficient $\alpha_1, \alpha_2, \alpha_3$ assigned to the three unfolded matrices may affect the performance of data recovery, in our evaluation, we consider the following versions of Algorithm 2:

- **Algorithm 2 with single unfolding:** only one unfolded matrix is assigned a positive weight 1 while the other two ones are assigned weight 0.
- **Algorithm 2 with double unfolding:** two of the unfolded matrices are assigned with equal weight 0.5;
- **Algorithm 2 with differentiation:** Divide the index set of all missing entries $\mathbf{\Theta}$ into two subsets:

  $$\mathbf{\Theta}_A = \{(i,j) | \mathcal{M}_{ijt} \text{ is known for at least one } t \in \{1, \ldots, T-1\}\},$$

  $$\mathbf{\Theta}_B = \{(i,j) | \mathcal{M}_{ijt} \text{ is missing for all } t \in \{1, \ldots, T-1\}\}.$$

  To recover the missing entries in $\mathbf{\Theta}_A$, apply Algorithm 2 with weights $\alpha_1 = \alpha_2 = 0, \alpha_3 = 1$. To recover the missing entries in $\mathbf{\Theta}_B$, apply Algorithm 2 with weights $\alpha_1 = 1, \alpha_2 = \alpha_3 = 0$.

We compare the above versions of Algorithm 2 with static prediction methods based on single frames, including Algorithm 1, DMFSGD and Vivaldi (7D). All versions of Algorithm 2 and Algorithm 1 are applied with feature extraction.

First, in Fig. 3.6(a) and Fig. 3.6(b), we compare Algorithm 2 with all the static prediction algorithms. For both low and high sample rates $R = 0.3$ and $R = 0.7$, Algorithm 2 leveraging tensor properties significantly outperforms the static latency prediction methods. It verifies the significant benefit of utilizing multi-frames, and reveals the strong correlation between different latency frames over time. By exploiting the low-rank structure of all three unfolded matrices, Algorithm 2 takes full advantage of the implicit information in the tensor data.

Second, we compare the performance of all different versions of Algorithm 2 in Fig. 3.6(c), Fig. 3.6(d), Fig. 3.6(e) and Fig. 3.6(f), under different weight assignment schemes for the unfolded matrices $\mathbf{F}_{(l)}$ for $l = 1, 2, 3$.

Fig. 3.6(c) and Fig. 3.6(d) compare various single unfolding schemes to Algorithm 2 with differentiation. Among all single unfolding schemes, Algorithm 2 performs similarly for $l = 1$ and 2, which outperforms $l = 3$. The reason is that if an entry is missing in all 3 frames, we cannot hope to recover it only based on $\mathbf{F}_{(3)}$.

The discrepancy between using the single unfolding $\mathbf{F}_{(1)}$ (or $\mathbf{F}_{(2)}$) and using $\mathbf{F}_{(3)}$ is shrinking when the sample rate is high ($R = 0.7$), because the chance that a node pair is missing in all 3 frames is small. This motivates us that we can benefit more from historical values of $\mathbf{M}_{i,j}$ when they are available rather than using network condition correlations between different nodes for estimation, and weight differentiation in Algorithm 2 would improve the recovery performance of our algorithm.

We further evaluate the performance of Algorithm 2 with double unfolding, and show the results in Fig. 3.6(e) and Fig. 3.6(f). The weight assignments used for double unfolding are $\alpha_1 = 0.5, \alpha_2 = 0, \alpha_3 = 0.5$. As we can see, the algorithm with differentiation still outperforms the algorithm that minimizes the sum of the ranks of two unfolded matrices, at both high ($R = 0.7$) and low ($R = 0.3$) sample rates.

Through all the above comparisons, we show the benefits of incorporating multiple latency frames to perform multi-frame recovery, and the advantage of differentiated treatments to missing node pairs $(i,\ j) \in \mathbf{\Theta}_A$ and $(i,\ j) \in \mathbf{\Theta}_B$. Specifically, the third unfolded matrix $\mathbf{F}_{(3)}$ is suitable for dealing with node pairs $(i,\ j) \in \mathbf{\Theta}_A$, while any of the first two unfolded matrices $\mathbf{F}_{(1)}$ and $\mathbf{F}_{(2)}$ are better to handle missing entries $(i,\ j) \in \mathbf{\Theta}_B$. It is shown that Algorithm 2 with such differentiation is optimal.

3.7 Proof for Lemma 3.3.1

In this appendix, we present our proof of Lemma 3.3.1, the key lemma in the proof of convergence of Algorithm 1. Before our formal proof, we firstly introduce some notations and preliminaries for matrix inequalities, which play an important role in our proof for Lemma 3.3.1.

We firstly introduce the theory of *majorization*, one of the most powerful techniques for deriving inequalities. Given a real vector $\mathbf{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$, we rearrange its components as $x_{[1]} \geq x_{[2]} \geq \ldots \geq x_{[n]}$. The definition of majorization is as follows. For $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$, if

$$\sum_{i=1}^{k} x_{[i]} \leq \sum_{i=1}^{k} y_{[i]} \text{ for } k = 1, \ldots, n-1$$

and

$$\sum_{i=1}^{n} x_{[i]} = \sum_{i=1}^{n} y_{[i]},$$

then we say that $\mathbf{x}$ is *majorized* by $\mathbf{y}$ and denote $\mathbf{x} \prec \mathbf{y}$. If

$$\sum_{i=1}^{n} x_{[i]} \leq \sum_{i=1}^{n} y_{[i]},$$

we say that $\mathbf{x}$ is *weakly majorized* by $\mathbf{y}$ and denote $\mathbf{x} \prec_w \mathbf{y}$. We introduce some properties for majorization and weak majorization, which can be found in quite a wide range of literature, e.g. [96]. Interested readers can find detailed proofs in these references.

Lemma 3.7.1 (cf. [96], Ch. 1)**.** *Let $g(t)$ be an increasing and convex function. Let $g(\mathbf{x}) := (g(x_1), g(x_2), \ldots, g(x_n))$ and $g(\mathbf{y}) := (g(y_1), g(y_2), \ldots, g(y_n))$. Then, $\mathbf{x} \prec_w \mathbf{y}$ implies $g(\mathbf{x}) \prec g(\mathbf{y})$.*

Theorem 3.7.1 (cf. [96], Ch. 9)**.** *If $\mathbf{A}$ is a Hermitian matrix (real symmetric for a real matrix $\mathbf{A}$), then we have $d(\mathbf{A}) \prec \lambda(\mathbf{A})$.*

Note that the singular values of $\mathbf{A}$ are the eigenvalues of the positive semidefinite matrix $\mathbf{A}^\top \mathbf{A}$. We then have:

Corollary 3.7.1. *If $\mathbf{A}$ is a real symmetric matrix, and we denote $|\mathbf{A}|$ as the positive semidefinite square root of $\mathbf{A}^\top \mathbf{A}$, we have $d(|\mathbf{A}|) \prec \lambda(|\mathbf{A}|) = \sigma(\mathbf{A})$.*

Lemma 3.7.2 (cf. [96], Ch. 9)**.** *For any matrices $\mathbf{A}$ and $\mathbf{B}$, we have $\sigma(\mathbf{AB}) \prec_w \sigma(\mathbf{A}) \circ \sigma(\mathbf{B})$, where $\circ$ denotes the Hadamard product (or entry-wise product).*

Lemma 3.7.3 (Abel's Lemma)**.** *For two sequences of real numbers $a_1, \ldots, a_n$ and $b_1, \ldots, b_n$, we have*

$$\sum_{i=1}^{n} a_i b_i = \sum_{i=1}^{n-1} (a_i - a_{i+1}) \left(\sum_{j=1}^{i} b_i \right) + a_n \sum_{i=1}^{n} b_i.$$

Lemma 3.7.4. *If $\mathbf{x} \prec \mathbf{y}$ and $\mathbf{w} = (w_1, w_2, \ldots, w_n)$, where $0 \leq w_1 \leq w_2 \leq \ldots \leq w_n$, we have*

$$\sum_{i=1}^{n} w_i x_i \geq \sum_{i=1}^{n} w_i y_i.$$

Proof. For any $1 \leq k < n$, we have

$$\sum_{i=1}^{k} x_i \leq \sum_{i=1}^{k} y_i.$$

Then, since $w_k \leq w_{k+1}$, we have

$$(w_k - w_{k+1}) \sum_{i=1}^{k} x_i \geq (w_k - w_{k+1}) \sum_{i=1}^{k} y_i$$

In addition, for $k = n$, we have

$$w_n \sum_{i=1}^{n} x_i = w_n \sum_{i=1}^{n} y_i,$$

since $x \prec y$ implies that the summation of x_i and y_i are identical. Summing up all n inequalities above, we have

$$\begin{aligned} & \sum_{k=1}^{n-1} (w_k - w_{k+1}) \left(\sum_{i=1}^{k} x_i \right) + w_n \sum_{i=1}^{n} x_i \\ \geq & \sum_{k=1}^{n-1} (w_k - w_{k+1}) \left(\sum_{i=1}^{k} y_i \right) + w_n \sum_{i=1}^{n} y_i. \end{aligned} \tag{3.20}$$

By applying the Abel's Lemma for both sides, we have

$$\sum_{k=1}^{n} w_k x_k \geq \sum_{k=1}^{n} w_k y_k,$$

which proves the lemma. □

Theorem 3.7.2 (cf. [96], Ch. 9)**.** *If* $\mathbf{A}$ *and* $\mathbf{B}$ *are two positive semidefinite matrices, then*

$$\operatorname{tr}(\mathbf{AB})^{\alpha} \leq \operatorname{tr}(\mathbf{A}^{\alpha}\mathbf{B}^{\alpha}), \quad \alpha > 1, \tag{3.21}$$

$$\operatorname{tr}(\mathbf{AB})^{\alpha} \geq \operatorname{tr}(\mathbf{A}^{\alpha}\mathbf{B}^{\alpha}), \quad 0 < \alpha \leq 1. \tag{3.22}$$

Now we are ready to prove our Lemma 3.3.1 as follows:

Proof of Lemma 3.3.1. The right inequality is a consequence of Lemma 3.7.2. To see this, let $g(t) = t^p$ for all $1 \leq p \leq 2$. For all $t \geq 0$, $g(t)$ is an increasing and convex funciton. Thus, we have $\sigma(\mathbf{AB}) \prec_w \sigma(\mathbf{A}) \circ \sigma(\mathbf{B})$ from Lemma 3.7.2, and from Lemma 3.7.1 we have $g(\sigma(\mathbf{AB})) \prec_w g(\sigma(\mathbf{A}) \circ \sigma(\mathbf{B}))$ and it implies that

$$\sum_{i=1}^{n} \sigma_i^p(\mathbf{AB}) \leq \sum_{i=1}^{n} \sigma_i^p(\mathbf{A}) \sigma_i^p(\mathbf{B}).$$

So now we can focus on the left inequality. Here we denote $|\mathbf{A}|$ as the positive semidefinite square root of $\mathbf{A}^{\top}\mathbf{A}$. Suppose the singular value decomposition of $\mathbf{A}$ is

$\mathbf{U}_A\mathbf{\Sigma}_A\mathbf{V}_A^\top$, and that of $\mathbf{B}$ is $\mathbf{U}_B\mathbf{\Sigma}_B\mathbf{V}_B^\top$. By the unitary invariance of the singular values and the Schatten-p norm, we have

$$||\mathbf{AB}||_p^p = ||\mathbf{U}_A\mathbf{\Sigma}_A\mathbf{V}_A^\top\mathbf{U}_B\mathbf{\Sigma}_B\mathbf{V}_B^\top||_p^p \tag{3.23}$$

$$= ||(\mathbf{\Sigma}_A\mathbf{V}_A^\top\mathbf{U}_B)\mathbf{\Sigma}_B||_p^p \tag{3.24}$$

$$= ||\mathbf{A}_1\mathbf{B}_1||_p^p. \tag{3.25}$$

Here we let $\mathbf{A}_1 := \mathbf{\Sigma}_A\mathbf{V}_A^\top\mathbf{U}_B$ and $\mathbf{B}_1 := \mathbf{\Sigma}_B$. Thus, without loss of generality, we can assume that $\mathbf{B}$ is diagonal. Then, from the definition of Schatten-p norm, we have

$$\begin{aligned}
||\mathbf{AB}||_p^p &= \mathrm{tr}(|\mathbf{AB}|^p) = \mathrm{tr}\left(\sqrt{\mathbf{B}^\top\mathbf{A}^\top\mathbf{AB}}\right)^p \\
&\geq \mathrm{tr}((\mathbf{B}^\top)^{\frac{p}{2}}(\mathbf{A}^\top\mathbf{A})^{\frac{p}{2}}\mathbf{B}^{\frac{p}{2}}) \qquad (3.26)\\
&= \mathrm{tr}((\mathbf{BB}^\top)^{\frac{p}{2}}(\mathbf{A}^\top\mathbf{A})^{\frac{p}{2}}) \\
&= \mathrm{tr}(|\mathbf{B}|^p(\mathbf{A}^\top\mathbf{A})^{\frac{p}{2}}) \\
&= \mathrm{tr}(|\mathbf{B}|^p|\mathbf{A}|^p) \qquad (3.27)
\end{aligned}$$

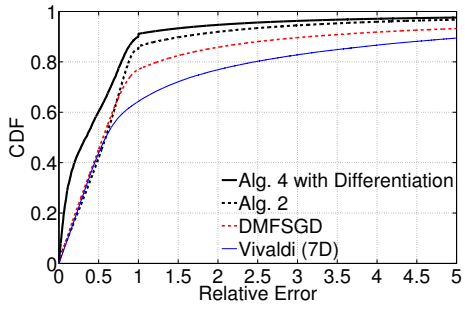
Here (3.26) is from (3.22) in Theorem 3.7.2, since $|\mathbf{B}|$ is diagonal with all nonnegative entries, and $\mathbf{A}^\top\mathbf{A}$ is a real symmetric matrix. Since $\mathbf{B}$ is diagonal, $d(|\mathbf{B}|)$ is just a permutation of its singular value vector $\sigma(\mathbf{B})$. Thus, we can simply rearrange the order of sum in (3.27) as

$$\begin{aligned}
\mathrm{tr}(|\mathbf{B}|^p|\mathbf{A}|^p) &= \sum_{i=1}^n d_i(|\mathbf{B}|)d_i(|\mathbf{A}|) \qquad (3.28)\\
&= \sum_{i=1}^n d_{[n-i+1]}(|\mathbf{B}|)d_{\pi(i)}(|\mathbf{A}|), \qquad (3.29)
\end{aligned}$$

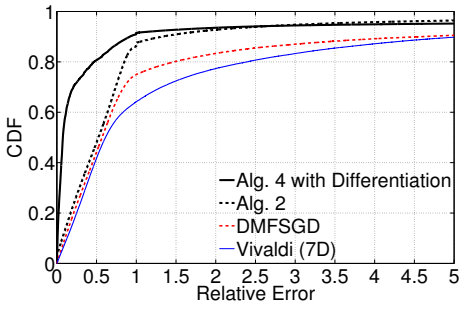
where $\pi(\cdot)$ is a permutation indicating the order of the new summation, and $d_{[i]}(|\mathbf{B}|) = \sigma_i(\mathbf{B})$. From Lemma 3.7.1, we can see that $d(|\mathbf{A}|) \prec \sigma(\mathbf{A})$, and by Lemma 3.7.4, we finally have

$$\begin{aligned}
||\mathbf{AB}||_p^p &= \sum_{i=1}^n d_{[n-i+1]}^p(|\mathbf{B}|)d_{\pi(i)}^p(|\mathbf{A}|) \qquad (3.30)\\
&= \sum_{i=1}^n \sigma_{n-i+1}^p(\mathbf{B})d_{\pi(i)}^p(|\mathbf{A}|) \qquad (3.31)\\
&\geq \sum_{i=1}^n \sigma_{n-i+1}^p(\mathbf{B})\sigma_i^p(\mathbf{A}). \qquad (3.32)
\end{aligned}$$

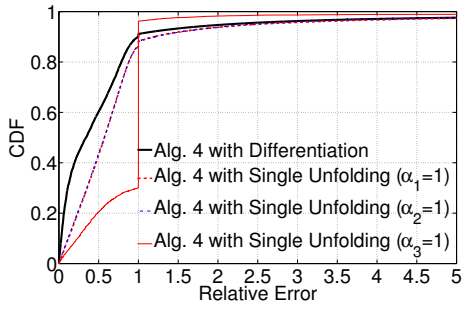
□



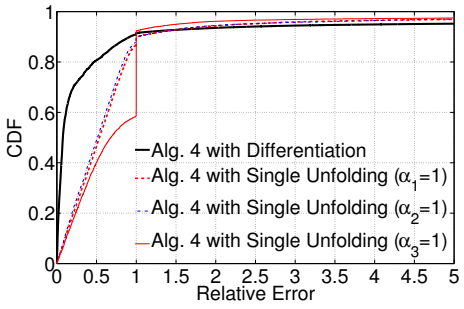
(a) Sample rate $R = 0.3$



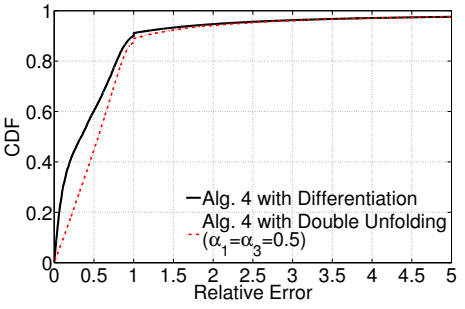
(b) Sample rate $R = 0.7$



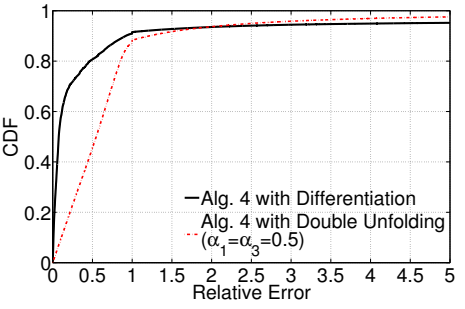
(c) Sample rate $R = 0.3$



(d) Sample rate $R = 0.7$



(e) Sample rate $R = 0.3$



(f) Sample rate $R = 0.7$

Figure 3.6: The CDFs of relative estimation errors on the missing values in the *current* frame with sample rates $R = 0.3$ and $R = 0.7$ for the Seattle dataset. Feature extraction has been applied in all experiments.

Chapter 4

Quantile Matrix Factorization

4.1 Background

In the last chapter, we use rank minimization approach, which is based on convex optimization techniques, to estimate a low-rank matrix from some known observations. In this chapter and Chapter 5, we use matrix factorization that doing low-rank matrix estimation that has been implemented and deployed in large-scale industrial machine learning systems. Although matrix factorization problems are nonconvex, they can be solved efficiently in practice at a large scale by several standard iterative optimization methods such as alternating minimization and stochastic gradient descent [14].

Nevertheless, a common limitation of all existing matrix estimation studies is that they aim to estimate a *conditional mean* for each pair and have ignored the fact that in many scenarios, i.e., quality of service (QoS) estimation including both latencies and throughput, may be highly skewed in reality.

For example, in a dataset made available by recent studies [50], we can observe that most web service response times are within 100 milliseconds while a few outliers could exceed one or even several seconds due to network congestion or temporary service unavailability [97, 98]. In this case, state-of-the-art matrix factorization techniques, which aim to minimize a mean squared error, tend to only explain the conditional mean response time or throughput between a user and a web service, which may deviate from the "most probable" value due to the existence of outliers. Therefore, using conditional mean QoS estimates to select and recommend top web services for each user may lead to significant biases.

In this chapter, we propose *Quantile Matrix Factorization* (QMF). Our contributions are with two folds: *First*, we propose the concept of *Quantile Matrix*

Factorization by replacing the least squares objective function in traditional matrix factorization with an asymmetric loss function similar to that in Quantile Regression [99]. The new objective function enables us to estimate the conditional *median* metrics, which can better explain the central tendency of skewed data, or to estimate a certain *percentile* of the value of interest. *Second*, although the QMF problem has a nonsmooth quantile objective, we propose a simple yet efficient Iteratively Reweighted Least Squares (IRLS) algorithm to efficiently solve a smooth approximation of the QMF problem.

The results in Chapter 4 have been published in our INFOCOM paper [40].

4.2 Problem Description

For ease of representation, we focus on the scenario of QoS value estimation in this chapter. Suppose we have a set m users and a set of n web services. Let $\mathbf{M} \in \mathbb{R}^{m \times n}$ denote the matrix of QoS values between all the users and servers, where the entry $\mathbf{M}_{i,j}$ represents the latency (or throughput) between user i and service j. Assume we have observed the QoS value between some user-service pairs. Let Ω denote the set of all such measured (i, j) pairs. The problem is to infer the missing QoS entries in $\mathbf{M}$ only based on the partially observed values $\Omega = \{\mathbf{M}_{i,j} | \text{QoS value of } (i, j) \text{ has been observed}\}$.

One effective and popular method that has been successfully adopted in network latency estimation based on partial observations is matrix factorization (e.g., [43], [98]). Matrix factorization solves a nonconvex optimization problem, assuming that each user i has a latent feature vector $\mathbf{x}_i \in \mathbb{R}^r$ and each service j has a latent feature vector $\mathbf{y}_j \in \mathbb{R}^r$. Let $\mathbf{X} := [\mathbf{x}_1, \ldots, \mathbf{x}_m]^\top$ and $\mathbf{Y} := [\mathbf{y}_1, \ldots, \mathbf{y}_n]^\top$. Then, the matrix factorization problem is to find two such tall matrices $\mathbf{X}$ and $\mathbf{Y}$, with $r \ll \{m, n, |\Omega|\}$ by solving

$$\min_{\mathbf{X} \in \mathbb{R}^{m \times r}, \mathbf{Y} \in \mathbb{R}^{n \times r}} \sum_{(i,j) \in \Omega} \mathcal{L}(\mathbf{M}_{i,j}, \hat{\mathbf{M}}_{i,j}) \quad \text{s.t.} \quad \hat{\mathbf{M}} = \mathbf{X}\mathbf{Y}^\top.$$

The most commonly used loss function in matrix factorization is the squared loss $(\mathbf{M}_{i,j} - \mathbf{x}_i^\top \mathbf{y}_j)^2$, with which the problem is to minimize the mean squared error (MSE):

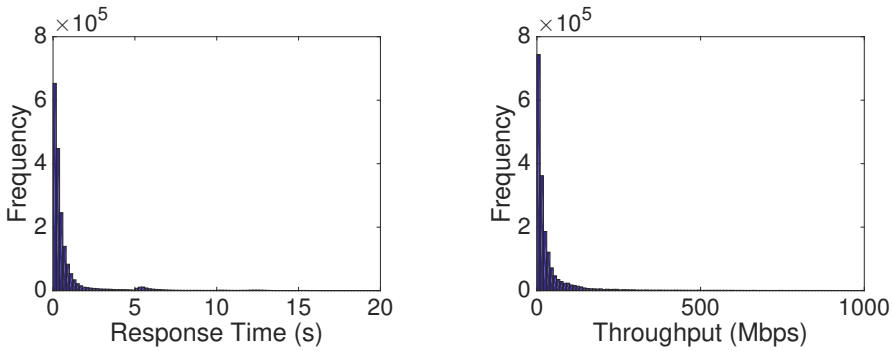
$$\begin{aligned} \min_{\mathbf{X},\mathbf{Y}} \quad & \frac{1}{2} \sum_{(i,j) \in \Omega} (\mathbf{M}_{i,j} - \hat{\mathbf{M}}_{i,j})^2 + \frac{\lambda}{2} \|\mathbf{X}\|_F^2 + \frac{\lambda}{2} \|\mathbf{Y}\|_F^2 \\ \text{s.t.} \quad & \hat{\mathbf{M}} = \mathbf{X}\mathbf{Y}^\top, \forall \mathbf{X} \in \mathbb{R}^{m \times r}, \mathbf{Y} \in \mathbb{R}^{n \times r}, \end{aligned} \tag{4.1}$$

where $\|\cdot\|_F$ denotes the matrix Frobenius norm. The term $\frac{\lambda}{2}\|\mathbf{X}\|_F^2$ and $\frac{\lambda}{2}\|\mathbf{Y}\|_F^2$ are usually called regularizer in order to encourage simple models to avoid over-fitting issues (which make the model fit observed entries well but drift away from unknown ones). Like linear regression, solving (4.1) actually aims to produce an optimal solution $\hat{\mathbf{M}}$ such that $\hat{\mathbf{M}}_{i,j} = \langle \mathbf{x}_i, \mathbf{y}_j \rangle$ estimates the *conditional mean* of the observation $\mathbf{M}_{i,j}$. For symmetric noise following a Gaussian distribution, the conditional mean is the most efficient estimator.

However, for skewed or heavy-tailed noises, the conditional mean can be far away from the central area where $\mathbf{M}_{i,j}$ is distributed, and thus is not representative of the underlying most frequent value of $\mathbf{M}_{i,j}$. In these cases, we need to develop new techniques to better characterize the median and tail behavior of observations, beyond conditional mean estimates.

To get an intuitive idea about the distributions of typical web service QoS data, in Fig. 4.1(a) and Fig. 4.1(b), we plot the response times and throughput values between 339 service users and 5825 web services distributed all around the world from a publicly available dataset [50]. Here, the response time is the duration between the time that a service user sends a request and the time that he/she has received the corresponding response, including both the service latency and network latency. We can see that both response times and throughput are highly skewed. In Fig. 4.1(a) we can see that 90% of measured response times are smaller than 1.7s, yet the largest measured latency is 19.9s. More importantly, the mean response time is 0.9s which is much larger than the median 0.32s, around which most response times are distributed. Similarly, in Fig. 4.1(b), the mean throughput is 47.56 kbps, while the median is only 13.95 kbps and 90% of measured throughput values are smaller than 103.70 kbps while the largest measured value is 1000 kbps. This also shows that the throughput is highly skewed and heavy-tailed.

Such a significant discrepancy between the mean and the median implies that the conventional matrix factorization minimizing mean squared error (MSE) may not be suitable for estimating web service QoS metrics, since it tends to yield a mean estimation conditioned on the observations, while in reality QoS data will most likely concentrate around their median values. Moreover, the use of MSE-based matrix factorization can not estimate a certain percentile of the QoS values.



(a) Response Time (b) Throughput

Figure 4.1: Histograms of response times and throughput between 5825 web services and 339 service users. Both QoS metrics are highly skewed.

4.2.1 From Quantile Regression to Quantile Matrix Factorization

We now extend the idea of quantile regression to the case of matrix factorization, and we call such a new approach *quantile matrix factorization*.

Quantile regression estimates conditional quantile functions. More formally, the τ-th quantile of a real-valued random variable Z is defined as

$$Q_\tau(\mathrm{Z}) = \inf\{z : \Pr(\mathrm{Z} \leq z) \geq \tau\},$$

among which the median is given by $Q_{0.5}(\cdot)$. If the random variable Z is conditional on some other variables or parameters, the quantile calculated here is called the *conditional quantile*.

A useful key observation [99] in quantile regression is

$$Q_\tau(\mathrm{Z}) = \arg\min_\beta \mathbb{E}[\rho_\tau(\mathrm{Z} - \beta)],$$

where

$$\rho_\tau(z) := z(\tau - \mathbb{1}(z < 0))$$

is called the *check loss function*, with $\mathbb{1}(z < 0)$ equal to 1 if $z < 0$ and equal to 0 otherwise. The rationale behind the above equation is that, for any $0 < \tau < 1$, minimizing $\rho_\tau(z)$ would push the lowest τ fraction to lie below z and $1 - \tau$ fraction to lie above z.

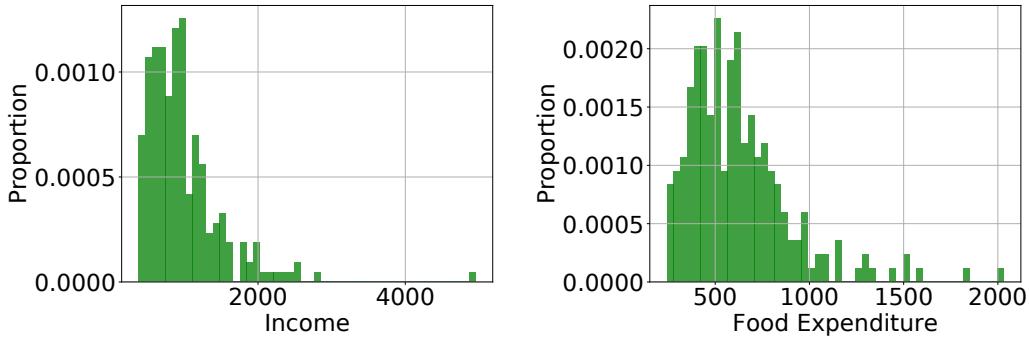
Given the above observation, we can estimate the quantile of a random variable Z from its sample observations $z_1, \ldots, z_N$ by solving the following optimization problem [99]:

$$\hat{Q}_\tau(\mathrm{Z}) = \arg\min_\beta \sum_{i=1}^{N} \rho_\tau(z_i - \beta). \tag{4.2}$$

Now we introduce a quantile-regression-like objective into the matrix factorization problem by replacing the squared loss in the original matrix factorization by a quantile check loss function. Under this setting, we can formulate the *Quantile Matrix Factorization* (QMF) problem as

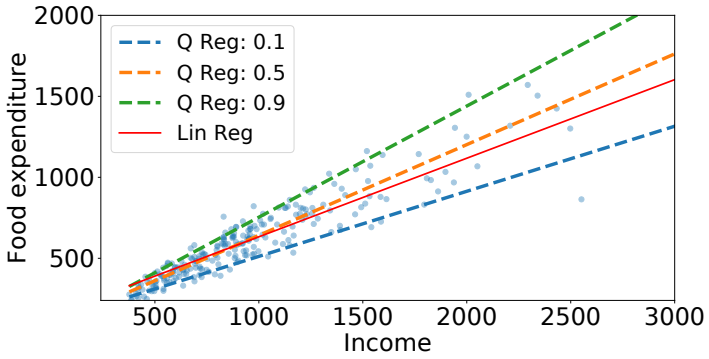
$$\begin{aligned} \min_{\mathbf{X},\mathbf{Y}} \quad & \sum_{(i,j)\in\Omega} \rho_\tau(\mathbf{M}_{i,j} - \hat{\mathbf{M}}_{i,j}) + \frac{\lambda}{2}\|\mathbf{X}\|_F^2 + \frac{\lambda}{2}\|\mathbf{Y}\|_F^2 \\ \text{s.t.} \quad & \hat{\mathbf{M}} = \mathbf{X}\mathbf{Y}^\top, \mathbf{X} \in \mathbb{R}^{m\times r}, \mathbf{Y} \in \mathbb{R}^{n\times r}, \end{aligned} \tag{4.3}$$

which aims to produce an optimal solution $\hat{\mathbf{M}}$ such that $\hat{\mathbf{M}}_{i,j} = \langle \mathbf{x}_i, \mathbf{y}_j \rangle$ estimates the τ-th *quantile* of the observation $\mathbf{M}_{i,j}$. When $\tau = 0.5$, we are essentially estimating the median of each unobserved $\mathbf{M}_{i,j}$.



(a) Income

(b) Food Expenditure



(c) Regression

Figure 4.2: Quantile regression vs. linear regression. By taking multiple quantile levels, we can have a more complete picture of distribution and provide better estimation than ordinary linear regression.

Here we take a toy example in Fig. 4.2 to show why quantile regression provides better estimation for skewed distribution. We investigate relationship between income and food expenditure for 235 working class households in 1857 Belgium[1]. From

[1]It is a built-in dataset in `statsmodels`, a Python statistics package: `https://www.statsmodels.`

Fig. 4.2(a) and Fig. 4.2(b), we can see that both income and food expenditure are with skewed distribution. Most values concentrate on the left side and some relatively large values also exist. As a result, many points are clustered on the left bottom in Fig. 4.2(c). We then perform ordinary linear regression and quantile regression on this dataset. The red solid line shows that linear regression can predict conditional mean, but the residual becomes larger when income increases. When doing quantile regression, we perform in three levels, namely, 0.1-th, 0.5-th and 0.9-th quantiles. We can see that the 0.5-th quantile, which estimates the conditional median, is closer to the center than linear regression. In addition, quantile regression at 0.1-th and 0.9-th levels can predict values for high income class more accurately.

From this example, we can see quantile regression is a powerful tool. In fact, it has been a well accepted method in economics, ecology and statistics. The above quantile-regression-like formulation has two unique strengths:

First, the QMF formulation in problem 4.3 can shift estimates from mean statistics to conditional medians or other quantiles of the observations. This is important in practice. The conventional MSE-based MF, as discussed above, is to minimize the mean squared error and actually estimates the mean of each $M_{i,j}$ given the observations. However, when the data is not Gaussian, this method is not guaranteed to be optimal, especially when the data are skewed or heavy-tailed.

Second, by focusing on different levels τ of quantiles, we can obtain a more complete picture of the data than the conventional MSE-based methods. Actually, the choice of τ depends on the practitioner's interests. If we are interested in the medians, we can set $\tau = 0.5$, and we can get the first and third quartiles by setting $\tau = 0.25$ and 0.75, respectively. If we are interested in the tail property, e.g., 10-percentile response time, we can set $\tau = 0.1$. In more complicated cases, we can even estimate a specific confidence interval for each $\mathbf{M}_{i,j}$, e.g., a 90% confidence interval of each $\mathbf{M}_{i,j}$ can be estimated by solving problem 4.3 under $\tau = 0.05$ to get a lower bound and then under $\tau = 0.95$ to get an upper bound.

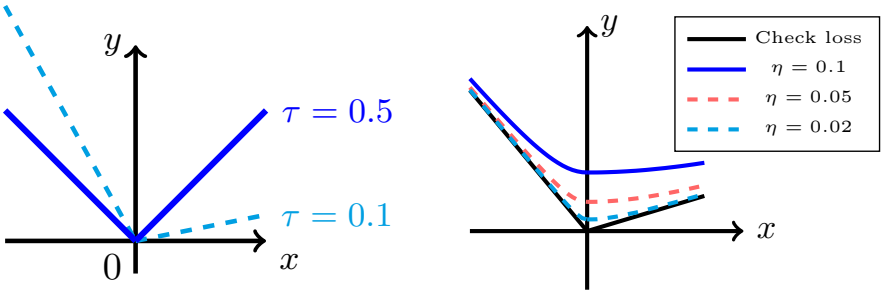
Because of these two strengths, the formulation in problem 4.3 is particularly suitable for our web service recommendation task. First, as shown in Fig. 4.1(a) and 4.1(b), the response time and throughput are highly skewed. Secondly, we can utilize QMF to help a user narrow down the search of web services by estimating extreme quantile values. QMF in our web service recommender system can also help

`org/dev/datasets/generated/engel.html`.

users exclude the worst services with long response time and low throughput, which can not be achieved by the traditional MSE-based estimator.

4.3 Algorithms

When it comes to directly optimizing the *nonsmooth* quantile loss function, choices of efficient algorithms are limited. Meanwhile, the nonsmooth characteristic also hinders its theoretical investigation. Many researchers use a smooth function to approximate the quantile loss function, e.g., [100]. In this thesis, we use the same strategy to find a smooth approximation to the quantile objective and then provide an Iteratively Reweighted Least Square (IRLS) algorithm to solve the smoothed approximation.



(a) The check loss $\rho_\tau(\cdot)$ (b) Smoothed Quantile Function

Figure 4.3: Smoothing a quantile check loss function. In both figures, we consider $\tau = 0.2$. (a) The check loss function in quantile regression, placing different weights on positive residuals and negative residuals. (b) An illustration of the smoothed quantile function.

We now introduce our smooth technique in Algorithm 3. For the residual $R_{i,j} := \mathbf{M}_{i,j} - \hat{\mathbf{M}}_{i,j}$, we have the following inequalities:

$$\rho_\tau(R_{i,j}) \leq \sqrt{\rho_\tau^2(R_{i,j}) + \eta^2} \leq \frac{1}{2}\left(\frac{\rho_\tau^2(R_{i,j}) + \eta^2}{z_{i,j}} + z_{i,j}\right)$$

for $\eta > 0, z_{i,j} > 0$. We notice that the constant η controls a trade-off between smoothness and approximation gap: a larger η can make the function smoother, but the gap between the first and the second term becomes larger, which may imply slower convergence rate. Therefore, we should carefully choose a η that is small enough but pertains some smoothness. Fig. 4.3 illustrates our proposed smooth technique for quantile check loss functions. From Fig. 4.3(b), we can clearly see how η controls the trade-off between smoothness and approximation gap. When $\eta = 0.1$,

Algorithm 3 IRLS for smoothed quantile matrix factorization.

Input: $\mathcal{P}_\Omega$, target rank r
Parameter: Smooth constant $\eta > 0$, maximum number of iterations T
Initialize $\mathbf{X}^{(0)}$ and $\mathbf{Y}^{(0)}$ with random values
for $t = 0$ to $T - 1$ **do**
    for $i = 1, \ldots, m$ **do**
        $\mathbf{x}_i^{(t+1)} \leftarrow \arg\min_{\mathbf{x} \in \mathbb{R}^r} h_i(\mathbf{x}; \mathbf{Y}^{(t)}, \mathbf{Z}^{(t)})$
    end for
    for $j = 1, \ldots, n$ **do**
        $\mathbf{y}_j^{(t+1)} \leftarrow \arg\min_{\mathbf{y} \in \mathbb{R}^r} h_j(\mathbf{y}; \mathbf{X}^{(t)}, \mathbf{Z}^{(t)})$
    end for
    $\mathbf{M}^{(t+1)} \leftarrow \mathbf{X}^{(t+1)} \mathbf{Y}^{(t+1)\top}$
    for $(i, j) \in \Omega$ **do**
        $R_{i,j}^{(t+1)} \leftarrow \mathbf{M}_{i,j} - \langle \mathbf{x}_i^{(t+1)}, \mathbf{y}_j^{(t+1)} \rangle$
        $z_{i,j}^{(t+1)} \leftarrow \sqrt{\rho_\tau^2(R_{i,j}^{(t+1)}) + \eta^2}$
    end for
end for
Output: $\mathbf{M} \leftarrow \mathbf{X}^{(T)} \mathbf{Y}^{(T)\top}$.

the blue solid line goes at the top and the gap from the check loss (the black solid line) becomes largest in all settings. For a lower η, we can see that the smooth function is getting closer to the check loss function while the function becomes less smooth.

For the first inequality, we can choose a small constant η related to the sample size $|\Omega|$, e.g., $\eta = 1/\log(|\Omega|)$. Then the gap between the first two terms will diminish as $|\Omega|$ increases. When $z_{i,j} = \sqrt{\rho_\tau^2(R_{i,j}) + \eta^2}$, the last term is minimized and the second inequality becomes the equality. Both the last two terms are smooth approximations of the quantile loss function and under certain conditions, the approximations could be as close as possible. Motivated by this fact, we solve the following problem as a smooth approximation of quantile matrix factorization:

$$\begin{aligned} \min_{\mathbf{X}, \mathbf{Y}} \quad & \sum_{(i,j) \in \Omega} \sqrt{\rho_\tau^2(\mathbf{M}_{i,j} - \hat{\mathbf{M}}_{i,j}) + \eta^2} + \frac{\lambda}{2} \|\mathbf{X}\|_F^2 + \frac{\lambda}{2} \|\mathbf{Y}\|_F^2 \\ \text{s.t.} \quad & \hat{\mathbf{M}} = \mathbf{X}\mathbf{Y}^\top, \forall \mathbf{X} \in \mathbb{R}^{m \times r}, \mathbf{Y} \in \mathbb{R}^{n \times r}. \end{aligned} \tag{4.4}$$

Considering the fact that both the number of users and web services can be very large in the real world, the optimization algorithm used should have simple steps in each iteration, with intermediate results that are easy to store. One very popular family of such algorithms is the Block-Coordinate Descent (BCD) method, which alternately minimizes each coordinate or variable, i.e., each row of $\mathbf{X}$ or $\mathbf{Y}$ will be

updated alternately in a sequential order or in parallel. However, since minimizing a square root loss function directly is not an easy task, we can try to minimize an approximate function instead. Under some conditions, minimizing an approximate function can even have performance similar to minimizing the original loss function. The Iteratively Reweighted Least Square (IRLS) algorithm [101] is such an example, and has recently been extended to a framework called block successive upper-bound minimization (BSUM) [102].

Motivated by IRLS and BSUM, we propose an algorithm that iteratively minimizes the following two functions for each user i and web service j:

$$h_i(\mathbf{x}; \mathbf{Y}, \mathbf{Z}) := \frac{1}{2} \sum_{j \in \Omega(i)} \frac{\rho_\tau^2(\mathbf{M}_{i,j} - \langle \mathbf{x}, \mathbf{y}_j \rangle)}{z_{i,j}} + \frac{\lambda}{2} \|\mathbf{x}\|_2^2,$$
$$h_j(\mathbf{y}; \mathbf{X}, \mathbf{Z}) := \frac{1}{2} \sum_{i \in \Omega(j)} \frac{\rho_\tau^2(\mathbf{M}_{i,j} - \langle \mathbf{x}_i, \mathbf{y} \rangle)}{z_{i,j}} + \frac{\lambda}{2} \|\mathbf{y}\|_2^2,$$

where $\Omega(i)$ denotes the set of observed services for user i, $\Omega(j)$ denotes the set of observed users for service j, and $\|\cdot\|_2$ denotes the ℓ_2 norm. The full procedure of our algorithm is shown in Algorithm 3. Note that according to Theorem 2 in [102], we can conclude that Algorithm 3 can converge to stationary points.

Now we present our approach to solve the above two sub-problems (shown in Steps 6 and 9) by rewriting them as Quadratic Programming (QP) problems. Then, we can use some standard solvers to solve QP. For simplicity, we only rewrite Step 6.

Suppose we have observed l web services for user i. For residual $R_{i,j}$, we can extract its positive component $R_{i,j}^+ := \max(R_{i,j}, 0)$ and its negative component $R_{i,j}^- := -\min(R_{i,j}, 0)$. Then, we can rewrite the corresponding term in $h_i(\mathbf{x}; \mathbf{Y}, \mathbf{Z})$ as

$$\frac{\rho_\tau^2(R_{i,j})}{z_{i,j}} = \tau^2 (R_{i,j}^+)^2 / z_{i,j} + (1-\tau)^2 (R_{i,j}^-)^2 / z_{i,j} \tag{4.5}$$
$$= \tau^2 (S_{i,j}^+)^2 + (1-\tau)^2 (S_{i,j}^-)^2, \tag{4.6}$$

where we denote $S_{i,j}^+ := R_{i,j}^+ / \sqrt{z_{i,j}}$ and $S_{i,j}^- := R_{i,j}^- / \sqrt{z_{i,j}}$. All such $S_{i,j}^+$ and $S_{i,j}^-$ form vectors $\mathbf{s}^+ \in \mathbb{R}_+^l$ and $\mathbf{s}^- \in \mathbb{R}_+^l$, respectively. We then denote $b_j = \mathbf{M}_{i,j} / \sqrt{z_{i,j}}$ and $\mathbf{y}'_j = \mathbf{y}_j / \sqrt{z_{i,j}}$ for $j \in \Omega(i)$. Then, we have

$$S_{i,j}^+ - S_{i,j}^- = (\mathbf{M}_{i,j} - \langle \mathbf{x}, \mathbf{y}_j \rangle) / \sqrt{z_{i,j}} = b_j - \langle \mathbf{x}, \mathbf{y}'_j \rangle.$$

We can finally convert the sub-problem in Step 6 into the following QP problem:

$$\begin{aligned} \min_{\mathbf{u},\mathbf{s}^+,\mathbf{s}^-} \quad & \tau^2\|\mathbf{s}^+\|_2^2 + (1-\tau)^2\|\mathbf{s}^-\|_2^2 + \frac{\lambda}{2}\|\mathbf{x}\|_2^2 \\ \text{s.t.} \quad & s_j^+ - s_j^- = b_j - \mathbf{y}_j'^{\top}\mathbf{u}, \forall j \in \Omega(i), \\ & \mathbf{x} \in \mathbb{R}^r, \mathbf{s}^+, \mathbf{s}^- \in \mathbb{R}_+^l. \end{aligned} \tag{4.7}$$

4.4 Convergence Analysis

We will show in this section that our proposed IRLS algorithm 3 can converge to stationary point. Our results are based on a generic result in [102], which proposes a block successive upper-bound minimization (BSUM) algorithm. We will take a brief introduction of this more generic algorithm, and then introduce our convergence analysis based on results in BSUM.

The BSUM algorithm is trying to find a solution for the following problem:

$$\begin{aligned} \min \quad & f(x_1, \ldots, x_N) \\ \text{s.t.} \quad & x_j \in \mathcal{X}_j. \end{aligned}$$

Here we denote $\mathbf{x} = (x_1, \ldots, x_N)$ and the feasible set $\mathcal{X} = \mathcal{X}_1 \times \ldots \times \mathcal{X}_N$ with $\mathcal{X}_j \subseteq \mathbb{R}^{d_j}$.

Algorithm 4 Pseudocode of the BSUM Algorithm [102, 103].

1: Find a feasible point $\mathbf{x}^0 \in \mathcal{X}$ and set $t = 0$.
2: **repeat**
3: Pick index set $\mathcal{I}^t$.
4: Let $x_j^t = \text{argmin}_{x_j \in \mathcal{X}_j} u_j(x_j, \mathbf{x}^{t-1})$, $\forall j \in \mathcal{I}^t$.
5: Set $x_k^t = x_k^{t-1}, \quad \forall k \notin \mathcal{I}^t$.
6: $t \leftarrow t + 1$.
7: **until** stops

We display the BSUM algorithm in Algorithm 4. Simply, the BSUM algorithm successively optimize some upper bounds or surrogate functions, instead of the original objectives which might be hard to do so. There are a lot of candidate upper bound functions $u_j(x_j, \mathbf{x}^{t-1})$, provided that the following assumptions are met:

Assumption 4.1. *The update function $u_j(x_j, \mathbf{x})$ satisfies the following conditions:*

$$u_j(x_j, \mathbf{x}) = f(\mathbf{x}) \quad \forall \mathbf{x} \in \mathcal{X}, \forall i, \tag{4.8}$$

$$u_j(x_j, \mathbf{y}) \geq f(x_j, \mathbf{y}_{-j}), \quad \forall x_j \in \mathcal{X}_j, \forall \mathbf{y} \in \mathcal{X}, \forall j, \tag{4.9}$$

$$u_j'(x_j, \mathbf{y}; d_j)|_{x_j = y_j} = f'(\mathbf{y}; \mathbf{d}),$$
$$\forall \mathbf{d} = (0, \ldots, d_j, \ldots, 0) \text{ s.t. } y_j + d_j \in \mathcal{X}_j, \forall j, \tag{4.10}$$

$$u_j(x_j, \mathbf{y}) \text{ is continuous in } (x_j, \mathbf{y}), \quad \forall j. \tag{4.11}$$

Here we denote $\mathbf{x}_{-j}$ to denote the vector $\mathbf{x}_{-j} := (x_1, \ldots, x_{j-1}, x_{j+1}, \ldots, x_M)$ that drops the j-th coordinate of $\mathbf{x}$. To select an index set for $\mathcal{I}^t$ at iteration t, we can either choose the cyclic rule, which chooses the coordinates cyclically in the order of "$1, 2, \ldots, N, 1, 2, \ldots$", or the randomized rule that picks some coordinates with a given probability distribution. We refer interested readers to [102, 103] for more details.

We now formally introduce the convergence result for the BSUM framework as follows:

Theorem 4.4.1 (Theorem 2, [102]). *1. Suppose that the function $u_j(x_j, \mathbf{y})$ is quasi-convex in x_j for $j = 1, \ldots, N$ and Assumption 4.1 holds. Furthermore, assume that the subproblem that minimizes $u_j(x_j, \mathbf{x}^{t-1})$ has a unique solution for any point $\mathbf{x}^{t-1} \in \mathcal{X}$. Then, every limit point $\mathbf{z}$ of the iterates generated by the BSUM algorithm is a coordinate-wise minimum of $f(\mathbf{x})$. In addition, if $f(\cdot)$ is regular at $\mathbf{z}$, then $\mathbf{z}$ is a stationary point of $f(\mathbf{x})$.*

2. Suppose the level set $\mathcal{X}^0 = \{\mathbf{x} | f(\mathbf{x}) \leq f(\mathbf{x}^0)\}$ is compact and Assumption 4.1 holds. Furthermore, assume that $f(\cdot)$ is regular at any point in $\mathcal{X}^0$ and the subproblem that minimizes $u_j(x_j, \mathbf{x}^{t-1})$ has a unique solution for any point $\mathbf{x}^{t-1} \in \mathcal{X}$ for at least $M - 1$ blocks. Then, the iterates generated by the BSUM algorithm converge to the set of stationary points, i.e.,

$$\lim_{t \to \infty} d(\mathbf{x}^t, \mathcal{X}^*) = 0,$$

where the distance function $d(\mathbf{x}, \mathcal{S})$ between $\mathbf{x}$ and a set $\mathcal{S} \subseteq \mathbb{R}^d$ is defined as

$$d(\mathbf{x}, \mathcal{S}) = \inf_{\mathbf{s} \in \mathcal{S}} \|\mathbf{x} - \mathbf{s}\|.$$

Now we apply Theorem 4.4.1 to obtain our convergence result. Note that we alternatively minimizing $\mathbf{x}_i$'s and $\mathbf{y}_j$'s by fixing $\mathbf{Y}$ and $\mathbf{X}$, respectively. Therefore, our objective function can be written as $\mathcal{F}(\mathbf{X}, \mathbf{Y}) := f(\mathbf{x}_1, \ldots, \mathbf{x}_m, \mathbf{y}_1, \ldots, \mathbf{y}_n)$ with $m + n$ blocks, where $\mathcal{F}(\mathbf{X}, \mathbf{Y})$ is the objective in (4.4). From (4.3), we can see that $h_i(\mathbf{x}_i; \mathbf{Y}, \mathbf{Z})$ and $h_j(\mathbf{y}_j; \mathbf{X}, \mathbf{Z})$ satisfy Assumption 4.1. Note that minimizing $h_i(\mathbf{x}_i; \mathbf{Y}, \mathbf{Z})$ or $h_j(\mathbf{y}_j; \mathbf{X}, \mathbf{Z})$ can be converted to a QP shown in (4.7), in which the quadratic coefficient matrix $\mathbf{Q}$ is a diagonal matrix with values of $\tau, 1 - \tau, \lambda/2$, so $\mathbf{Q}$ should be positive definite and we can conclude that the subproblem (4.7) has a unique solution. Therefore, we have our proposed IRLS can converge to coordinatewise minimum. Note that $\mathcal{F}(\mathbf{X}, \mathbf{Y})$ is smooth for all $\eta > 0$, $\mathcal{F}$ is regular at $(\mathbf{X}^{(t)}, \mathbf{Y}^{(t)})$ for all t, so we guarantee that the limit point of $\mathcal{F}(\mathbf{X}, \mathbf{Y})$ is a stationary point.

Table 4.1: Ranking Performance Comparison of Response Time on NDCG@k and Precision@k (Larger value indicates higher ranking performance). Here N@k indicates NDCG@k and P@k indicates Precision@k

	Metric	QMF, 0.1	QMF, 0.25	QMF, 0.5	PMF
Sampling Rate 1%	N@10	0.217	0.354	**0.409**	0.135
	N@100	0.341	0.42	**0.453**	0.13
	P@10	0.008	0.016	**0.017**	0.013
	P@100	0.107	0.165	**0.184**	0.036
Sampling Rate 10%	N@10	0.572	**0.615**	0.595	0.085
	N@100	0.638	0.694	**0.715**	0.165
	P@10	0.036	**0.042**	0.037	0.013
	P@100	0.391	0.473	**0.52**	0.044
Sampling Rate 30%	N@10	0.568	**0.592**	0.563	0.059
	N@100	0.65	0.703	**0.713**	0.212
	P@10	0.04	0.032	**0.042**	0.018
	P@100	0.413	0.49	**0.523**	0.062

4.5 Performance Evaluation

We evaluate our QMF method on a publicly available dataset, which contains response time and throughput records between 339 users and 5825 web services distributed worldwide, made available by a previous study [50].

4.5.1 Experimental Setup

We will first evaluate the performance of QMF in comparison to existing web service recommendation schemes in terms of two performance metrics, namely, *NDCG@k*

Table 4.2: Ranking Performance Comparison of Throughput on NDCG@k and Precision@k (Larger value indicates higher ranking performance). Here N@k indicates NDCG@k and P@k indicates Precision@k

	Metric	QMF, 0.1	QMF, 0.25	QMF, 0.5	PMF
Sampling Rate 1%	N@10	0.272	0.548	**0.601**	0.02
	N@100	0.252	0.472	**0.586**	0.052
	P@10	0.016	0.059	**0.078**	0.005
	P@100	0.048	0.155	**0.222**	0.012
Sampling Rate 10%	N@10	0.442	0.65	**0.756**	0.041
	N@100	0.521	0.696	**0.779**	0.058
	P@10	0.017	0.033	**0.075**	0.01
	P@100	0.146	0.313	**0.429**	0.018
Sampling Rate 30%	N@10	0.586	0.743	**0.779**	0.079
	N@100	0.67	**0.779**	0.774	0.095
	P@10	0.026	0.067	**0.103**	0.016
	P@100	0.279	**0.429**	0.41	0.025

(Normalized Discounted Cumulative Gain at top k) and *Precision@k* (Precision at top k). These are two popular performance metrics for evaluating recommendation and ranking effectiveness. Instead of evaluating the relative/absolute pairwise prediction errors, these two metrics compare the gap between the *predicted order* and the *observed order*. Precision@k measures how many true top-k services in the observation are correctly predicted by an algorithm. Formally, let $\mathcal{P}_k(i)$ be the *predicted* set of top k services in terms of a QoS metric, and $\mathcal{P}_k^*(i)$ be the *observed* set, and Precision@$k(i)$ for user i is defined as

$$\text{Precision@}k(i) = \frac{1}{k} \sum_{j \in \mathcal{P}_k(i)} \mathbb{1}(j \in \mathcal{P}_k^*(i)).$$

Given a predicted ranked list of services π_i for user i, the NDCG@k metric is defined as

$$\text{NDCG@}k(i) := \frac{\text{DCG@}k(i, \pi_i)}{\text{DCG@}k(i, \pi_i^*)},$$

where

$$\text{DCG@}k(i, \pi_i) = rel(i, \pi_i(1)) + \sum_{j=2}^{k} \frac{rel(i, \pi_i(j))}{\log_2 j},$$

where the value $rel(i, \pi_i(j))$ is the relevance of the service $\pi_i(j)$. In our experiments, we simply use the observed QoS metrics as the relevance values.

We also evaluate the recovery accuracy of QMF in comparison to several state-of-the-art web service recommendation schemes, using the relative estimation errors

(REs) on missing entries, which are defined as $|\mathbf{M}_{i,j} - \hat{\mathbf{M}}_{i,j}|/\mathbf{M}_{i,j}$ for $(i,j) \notin \Omega$. In particular, we compare the following schemes:

- **Algorithm 3 (QMF)**: for each algorithm we set three quantiles $\tau = 0.1, 0.25, 0.5$, which represent the 10% quantile, the first quartile, and the median, respectively.
- **PMF (Probabilistic Matrix Factorization)**: a widely-used implementation of the matrix factorization model [55], which has been used to predict the response times of web services [104] with a loss function of MSE.

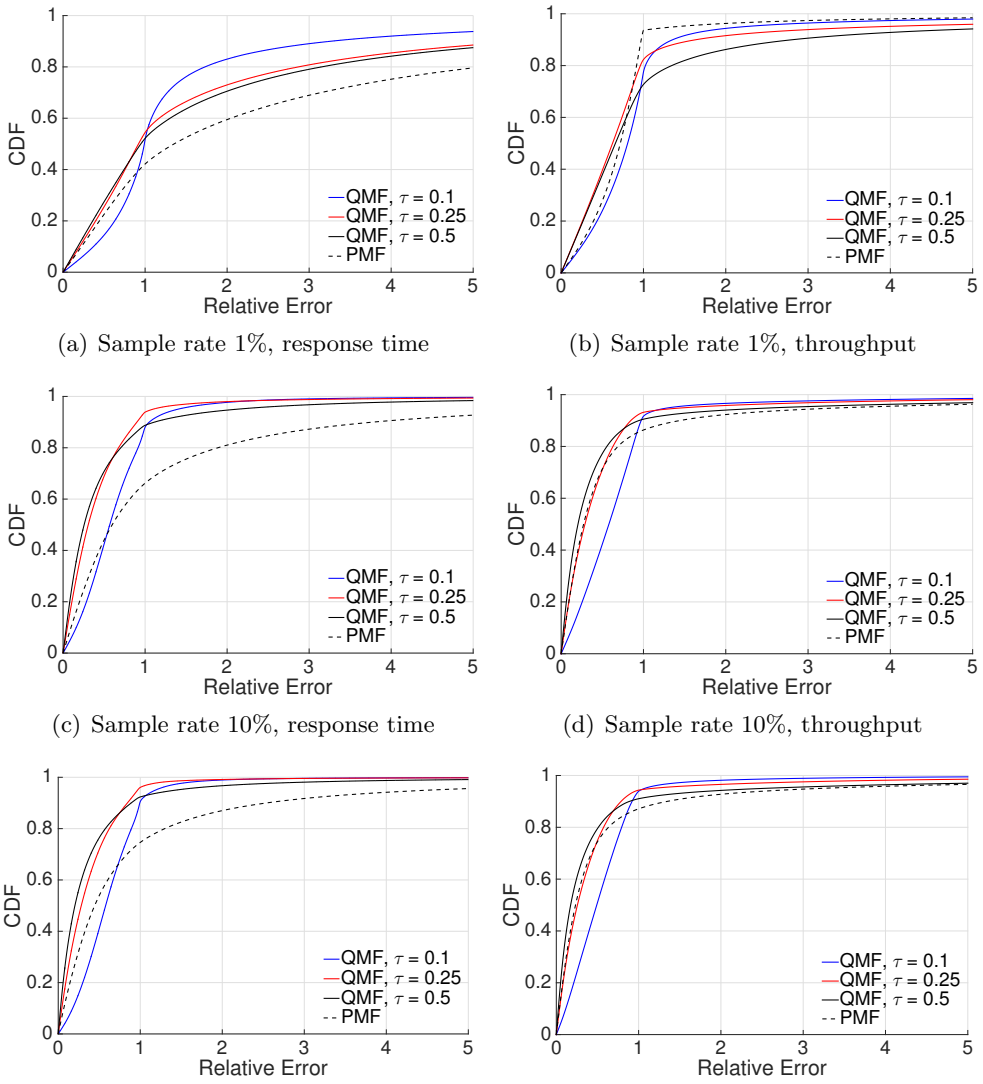
The user-service matrices in the real world are typically very sparse, since a user may have ever connected to only a small number of web services. We randomly choose a subset of observed values in the user-service matrix with different sampling rate, which is the ratio of the number of known entries in M to the number of all the entries. In particular, we randomly set 1%, 10% and 30% of the matrix entries as observed. Our algorithms and other baseline algorithms are employed to estimate the missing QoS values and predict the ranking of web services for each user in the descending order of the corresponding QoS values. For QMF, we set the dimension of latent feature vectors to $r = 10$, and the smooth constant $\eta = 1/\log(|\Omega|)$ as we have discussed in Sec. 4.2. For PMF, we also set the dimension of latent feature vectors as $r = 10$.

4.5.2 Ranking performance

Table 4.1 and 4.2 show the ranking performance in NDCG and precision of response time and throughput under the sampling rates of 1%, 10% and 30%. In this table we focus on four metrics: NDCG@10, NDCG@100, Precision@10 and Precision@100. We boldface the best performance for each column in the Table 4.1 and 4.2.

Compared with PMF, which minimizes MSE, Algorithm 3 obtains better prediction accuracy for both response time and throughput under all settings of τ. Since the data is highly skewed, our QMF algorithm can estimate different quantiles which are closer to the center of distribution. In particular, we observe highest NDCG and precision scores under $\tau = 0.5$ in most cases. This implies that estimating median is more robust than estimating mean.

For QMF, we can see that the ranking results under $\tau = 0.1$ are worse than results under $\tau = 0.25$ and $\tau = 0.5$, since the quantile $\tau = 0.1$ is more extreme



(a) Sample rate 1%, response time

(b) Sample rate 1%, throughput

(c) Sample rate 10%, response time

(d) Sample rate 10%, throughput

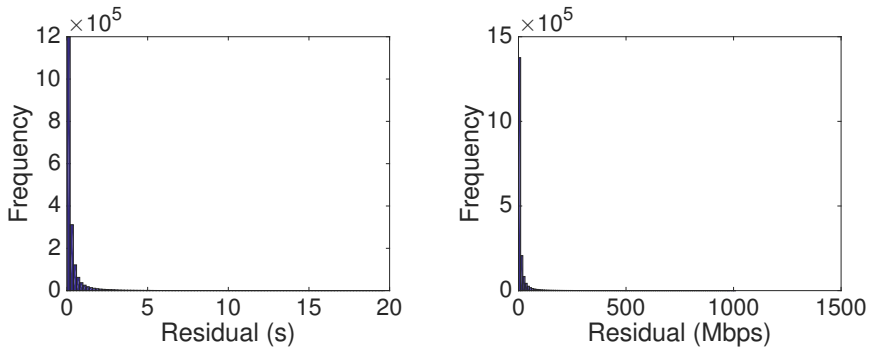
(e) Sample rate 30%, response time

(f) Sample rate 30%, throughput

Figure 4.4: The CDFs of relative estimation errors of response time and throughput on the missing values with sample rate 1%, 10% and 30%.

than $\tau = 0.25$ and $\tau = 0.5$ and the estimation under such quantile needs more data. When explicit user features and explicit service features are incorporated, they can provide initial estimation and it leads to decrease the demand of samples and lead to better performance.

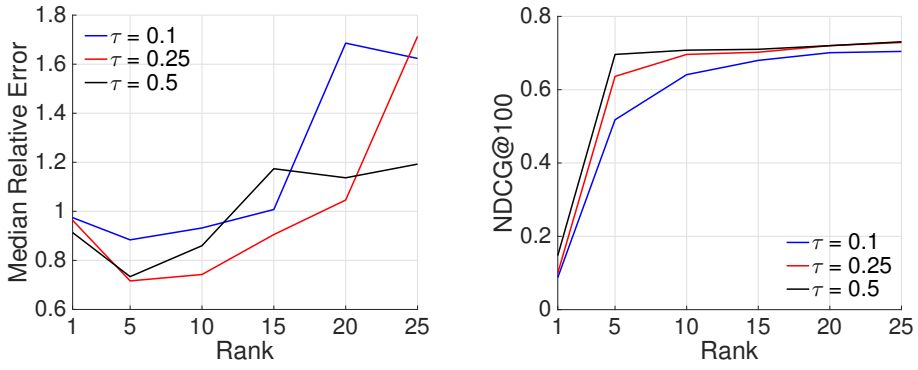
Now let's see why our QMF is better in terms of ranking. We did top $r = 10$ singular value decomposition (SVD) for both response time matrix and throughput matrix and we plot the results in Fig. 4.5. It is well known that top r SVD of M provides its best approximation of rank r in terms of Frobenius norm [105], which is actually the MSE. We plot the residuals of such method in Fig. 4.5. In these figures,



(a) Response Time (b) Throughput

Figure 4.5: Histograms of residuals via MSE minimization

90% of residuals of response time are smaller than 0.8, while the largest residual can be as large as 19.73. Also, 90% of residuals of throughput are smaller than 30.75, but the largest residual is 1011. And now it is clear to see that in these two datasets, the residuals are still highly skewed. Then we can conclude that if we use the conditional mean for ranking web services, the results are not accurate, because centers of these two datasets are distributed far away from their conditional means.



(a) Median Relative Error (b) NDCG@100

Figure 4.6: Impact of dimension of latent vectors on QMF in terms of median relative error and NDCG@100.

4.5.3 Recovery accuracy

We plot the relative estimation errors on missing response time and on missing throughput in Fig. 4.4 under three settings of QMF and PMF. In Fig. 4.4(a), 4.4(c) and 4.4(e), we can see that PMF is inferior to QMF under $\tau = 0.25$ and 0.5 because PMF is targeting minimizing MSE to estimate the conditional mean, but the highly skewness of response time distribution leads the mean to be far away from the

center. For QMF with $\tau = 0.1$, we can see that there are fewer small relative errors and large relative errors than others, and these errors concentrate on a small range, especially under low sampling rate as shown in Fig. 4.4(a). For the throughput results in Fig. 4.4(b), 4.4(d) and 4.4(f), our algorithm is only slightly better than PMF.

4.5.4 Impact of the latent feature dimension

We further investigate the impact of r, the dimension of user and service latent feature vectors. In this experiment, we set the sampling rate to be 10% and compare the median of relative estimation errors, called *median relative error*, and the NDCG@100 score. We test the impact of the dimension r on QMF for the response time dataset, and plot the results in Fig. 4.6. We can clearly see that both the ranking precision and the estimation precision increase as the latent feature dimension increases. However, when $r > 15$, the ranking performance almost stops increasing. In addition, a higher dimension introduces higher computational cost. Therefore, we suggest that r should be set in the range 5–15. The trend shown in these two figures also holds for other ranking metrics, e.g., precision$k$, and for other estimation error metrics, e.g., mean relative errors. The results on the throughput dataset are similar and omitted due to the space limit.

Chapter 5

Expectile Matrix Factorization

5.1 Background

In the last chapter, we have seen how QMF can handle the challenges brought by skewed and heavy-tailed data that are prevalent in the real world in order to achieve robustness to outliers and to better interpret the central tendency or dispersion of observations. In this chapter, we propose the concept of *expectile matrix factorization (EMF)* by replacing the symmetric least squares loss function in conventional matrix factorization with a loss function similar to those used in expectile regression [106]. Our scheme is different from weighted matrix factorization [107], in that we not only assign different weights to different residuals, but assign each weight *conditioned on whether the residual is positive or negative.* Intuitively speaking, our expectile matrix factorization problem aims to produce a low-rank matrix $\hat{\mathbf{M}}$ such that $\mathcal{A}(\hat{\mathbf{M}})$ can estimate any ωth conditional expectiles of the observations, not only enhancing the robustness to outliers, but also offering more sophisticated statistical understanding of observations from a matrix beyond mean statistics.

We investigate expectile matrix factorization from several aspects in this chapter. *First*, we propose an efficient algorithm based on alternating minimization and quadratic programming to solve expectile matrix factorization, which has low complexity similar to that of alternating least squares in conventional matrix factorization. *Second*, we theoretically prove that under certain conditions, expectile matrix factorization retains the desirable properties that without noise, it achieves the global optimality and exactly recovers the true underlying low-rank matrices. This result generalizes the prior result [57] regarding the optimality of alternating minimization for matrix estimation under the symmetric least squares loss (corresponding to $\omega = 0.5$ in EMF) to a general class of "asymmetric least squares" loss

functions for any $\omega \in (0, 1)$. The results are obtained by adapting a powerful tool we have developed on the theoretical properties of weighted matrix factorization involving varying weights across iterations. *Third*, for data generated from a low-rank matrix contaminated by skewed noise, we show that our schemes can achieve better approximation to the original low-rank matrix than conventional matrix factorization based on least squares. *Finally*, we also performed extensive evaluation based on a real-world dataset containing web service response times between 339 clients and 5825 web services distributed worldwide. We show that the proposed EMF saliently outperforms the state-of-the-art matrix factorization scheme based on least squares in terms of web service latency recovery from only 5-10% of samples.

The results in Chapter 5 have been published in our AAAI paper [41].

5.2 Expectile Matrix Factorization

In this chapter we focus on matrix sensing, a more general problem of matrix completion discussed in Chapter 3 and 4. Given a linear mapping $\mathcal{A} : \mathbb{R}^{m\times n} \to \mathbb{R}^p$, we can get p observations of an $m \times n$ matrix $\mathbf{M}^* \in \mathbb{R}^{m\times n}$. In particular, we can decompose the linear mapping $\mathcal{A}$ into p inner products, i.e., $\langle \mathbf{A}_i, \mathbf{M}^* \rangle$ for $i = 1, \ldots, p$, with $\mathbf{A}_i \in \mathbb{R}^{m\times n}$. Denote the p observations by a column vector $\mathbf{b} = (b_1, \ldots, b_p)^\top \in \mathbb{R}^p$, where b_i is the observation of $\langle \mathbf{A}_i, \mathbf{M}^* \rangle$ and may contain independent random noise. The *matrix sensing* problem is to recover the underlying true matrix $\mathbf{M}^*$ from observations $\mathbf{b}$, assuming that $\mathbf{M}^*$ has a low rank. Note that when $\mathbf{A}_i$ is a matrix that has only one nonzero entry with the value of 1 and vanishes to zero at all other entries, matrix sensing becomes the matrix completion problem.

We can use matrix factorization to solve this problem. Matrix factorization assumes that the matrix $\mathbf{M}^*$ has a rank no more than k, and can be factorized into two tall matrices $\mathbf{X} \in \mathbb{R}^{m\times k}$ and $\mathbf{Y} \in \mathbb{R}^{n\times k}$ with $k \ll \{m, n, p\}$. Specifically, it estimates $\mathbf{M}^*$ by solving the following nonconvex optimization problem:

$$\min_{\mathbf{X}\in\mathbb{R}^{m\times k}, \mathbf{Y}\in\mathbb{R}^{n\times k}} \sum_{i=1}^{p} \mathcal{L}(b_i, \langle \mathbf{A}_i, \mathbf{M} \rangle) \quad \text{s.t.} \quad \mathbf{M} = \mathbf{X}\mathbf{Y}^\top,$$

where $\mathcal{L}(\cdot, \cdot)$ is a loss function. We denote the optimal solution to the problem above by $\hat{\mathbf{M}}$.

The most common loss function used in matrix factorization is the squared loss $(b_i - \langle \mathbf{A}_i, \mathbf{X}\mathbf{Y}^\top \rangle)^2$, with which the problem is to minimize the mean squared error

(MSE):

$$\min_{\mathbf{X}\in\mathbb{R}^{m\times k},\mathbf{Y}\in\mathbb{R}^{n\times k}} \sum_{i=1}^{p} \frac{1}{2}(b_i - \langle \mathbf{A}_i, \mathbf{X}\mathbf{Y}^\top \rangle)^2. \tag{5.1}$$

Just like linear regression based on least squares, (5.1) actually aims to produce an $\hat{\mathbf{M}}$ which estimates the conditional mean of $\mathbf{M}^*$ given partial observations. For symmetric Gaussian noise, the conditional mean is the most efficient estimator. However, for skewed or heavy-tailed noise, the conditional mean can be far away from the central area where elements of the true $\mathbf{M}^*$ are distributed. In these cases, we need to develop new techniques to better characterize the central tendency, dispersion and tail behavior of observations, beyond mean statistics.

Similar to quantile regression, expectile regression [106] is also a regression technique that achieves robustness against outliers, while in the meantime is more computationally efficient than quantile regression by adopting a smooth loss function. In particular, suppose samples $\{(\mathbf{x}_i, y_i), i = 1, \ldots, n\}$ are generated from a linear model $y_i = \mathbf{x}_i^\top \boldsymbol{\beta}^* + \varepsilon_i$, where $\mathbf{x}_i = (1, x_{i1}, \ldots, x_{ip})^\top \in \mathbb{R}^{p+1}$ are predictors and $y_i \in \mathbb{R}$ is the response variable. The expectile regression estimates $\boldsymbol{\beta}^*$ by solving

$$\underset{\boldsymbol{\beta}}{\text{minimize}} \quad \sum_{i=1}^{n} \rho_\omega^{[2]}(y_i - \mathbf{x}_i^\top \boldsymbol{\beta}),$$

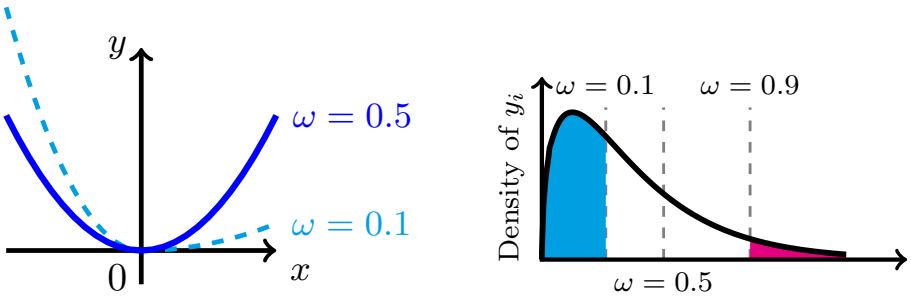
where for a chosen constant $\omega \in (0, 1)$, $\rho_\omega^{[2]}(\cdot)$ is the "asymmetric least squares" loss function given by

$$\rho_\omega^{[2]}(t) := t^2 \cdot |\omega - \mathbb{1}(t < 0)|,$$

where $\mathbb{1}(t < 0)$ is the indicator function such that it equals to 1 if $t < 0$ and 0 otherwise.

Fig. 5.1(a) illustrates the shape of $\rho_\omega^{[2]}(\cdot)$. When $\omega < 0.5$, we can see that the cost of a positive residual is lower than that of a negative residual, thus encouraging a smaller estimate $\hat{y}_i$ for the response variable, and vice versa when $\omega > 0.5$. This fact implies that when the response variable y_i are not Gaussian but highly skewed, we can choose an ω to push $\hat{y}_i$ to its most probable area (i.e., the mode or median) while being robust to outliers, as shown in Fig. 5.1(b).

We now extend expectile regression to the case of matrix estimation. Formally, define $r_i := b_i - \langle \mathbf{A}_i, \mathbf{X}\mathbf{Y}^\top \rangle$ as the residual for b_i. Then, in loss minimization, we weight each squared residual r_i^2 by either ω or $1 - \omega$, conditioned on whether it is positive or negative. Therefore, we formulate *expectile matrix factorization* (EMF)



(a) $\rho_\omega^{[2]}(\cdot)$ (b) Expectile Regression

Figure 5.1: (a) The asymmetric least squares loss function, placing different weights on positive residuals and negative residuals. (b) For a skewed χ_3^2 distribution, expectile regression with $\omega = 0.1$ generates an estimate closer to the mode than the conditional mean ($\omega = 0.5$) does due to the long tail.

as the following problem:

$$\min_{\mathbf{X}\in\mathbb{R}^{m\times k},\mathbf{Y}\in\mathbb{R}^{n\times k}} F(\mathbf{X},\mathbf{Y}) := \sum_{i=1}^{p} \rho_\omega^{[2]}(b_i - \langle \mathbf{A}_i, \mathbf{X}\mathbf{Y}^\top\rangle). \tag{5.2}$$

Apparently, the MSE-based approach (5.1) is a special case of problem (5.2) by setting $\omega = 0.5$, which places equal weights on both positive and negative residuals.

Note that expectile matrix factorization proposed above is different from weighted MSE [107], where a different yet fixed (predefined) weight is assigned to different residuals. In expectile matrix factorization, each weight is either ω or $1-\omega$, depending on whether the residual of the estimate is positive or negative, i.e., we do not know the assignments of weights before solving the optimization problem. In other words, problem (5.2) estimates an $\hat{\mathbf{M}}$ such that each $\langle \mathbf{A}_i, \hat{\mathbf{M}}\rangle$ estimates the ωth *conditional expectile* [106] of b_i. In the meantime, expectiles are based on second-order moments and thus it is feasible to solve EMF efficiently, which we show in the next section.

Just like expectile regression, the main attraction of expectile matrix factorization goes beyond robustness to outliers. Being able to estimate any ωth expectile of observations, EMF can characterize different measures of central tendency and statistical dispersion, and is useful to obtain a more comprehensive understanding of data distribution. For example, if we are interested in the tail behavior, we could set $\omega = 0.9$ and if we are interested in the conditional median in a highly skewed dataset, we could set $\omega < 0.5$.

5.3 Algorithm and Theoretical Results

We propose an efficient algorithm to solve expectile matrix factorization via a combined use of alternating minimization and quadratic programming, as shown in Algorithm 5, with complexity similar to that of alternating least squares in conventional matrix factorization. To better approach potential optimal solutions, we first sum up all measurement matrices $\mathbf{A}_i$ weighted by b_i, and perform Singular Value Decomposition (SVD) to get top k singular values.

Algorithm 5 Alternating minimization for expectile matrix factorization. In this algorithm, we use $\bar{X}$ to highlight that $\bar{X}$ is orthonormal.

1: **Input**: observations $\mathbf{b} = (b_1, \ldots, b_p)^\top \in \mathbb{R}^p$, measurement matrices $\mathbf{A}_i \in \mathbb{R}^{m \times n}$, $i = 1, \ldots, p$.
2: **Parameter**: Maximum number of iterations T
3: $(\bar{\mathbf{X}}^{(0)}, \mathbf{D}^{(0)}, \bar{\mathbf{Y}}^{(0)}) = \texttt{SVD}_k(\sum_{i=1}^p b_i \mathbf{A}_i)$ ▷ Singular Value Decomposition to get top k singular values
4: **for** $t = 0$ to $T - 1$ **do**
5: $\quad \mathbf{Y}^{(t+0.5)} \leftarrow \arg\min_{\mathbf{Y}} F(\bar{\mathbf{X}}^{(t)}, \mathbf{Y})$
6: $\quad \bar{\mathbf{Y}}^{(t+1)} \leftarrow \texttt{QR}(\mathbf{Y}^{(t+0.5)})$ ▷ QR decomposition
7: $\quad \mathbf{X}^{(t+0.5)} \leftarrow \arg\min_{\mathbf{X}} F(\mathbf{X}, \bar{\mathbf{Y}}^{(t+1)})$
8: $\quad \bar{\mathbf{X}}^{(t+1)} \leftarrow \texttt{QR}(\mathbf{X}^{(t+0.5)})$
9: **end for**
10: **Output**: $\mathbf{M}^{(T)} \leftarrow \mathbf{X}^{(T-0.5)} \bar{\mathbf{Y}}^{(T)\top}$

The QR decompositions in Step 6 and Step 8 are *not necessary* and are only included here to simplify the presentation of theoretical analysis. QR decomposition ensures the orthonormal property: given an orthonormal matrix $\mathbf{X}$ (or $\mathbf{Y}$), the objective function $F(\mathbf{X}, \mathbf{Y})$ is strongly convex and smooth with respect to $\mathbf{Y}$ (or $\mathbf{X}$), as shown in the appendix. However, it has been proved [108] that when $\omega = 0.5$, alternating minimization with and without QR decomposition are equivalent. The same conclusion also holds for all ω. Therefore, in performance evaluation, we do not have to and did not apply QR decomposition.

The subproblems in Step 5 and Step 7 can be solved efficiently with standard quadratic program (QP) solvers after some reformulation. We now illustrate such equivalence to QP for Step 5, which minimizes $F(\bar{\mathbf{X}}, \mathbf{Y})$ given $\bar{\mathbf{X}}$. Let $r_i^+ := \max(r_i, 0)$ denote the positive part of residual r_i, and $r_i^- := -\min(r_i, 0)$ denote the negative part of r_i. We have $r_i = r_i^+ - r_i^-$, and the asymmetric least squares loss can be rewritten as

$$\rho_\omega^{[2]}(r_i) = \omega (r_i^+)^2 + (1 - \omega)(r_i^-)^2.$$

Given $\bar{\mathbf{X}}$, we have

$$\mathcal{A}(\bar{\mathbf{X}}\mathbf{Y}^\top) = \{\langle \mathbf{A}_i, \bar{\mathbf{X}}\mathbf{Y}^\top \rangle\}_{i=1}^p = \{\langle \mathbf{A}_i^\top \bar{\mathbf{X}}, \mathbf{Y} \rangle\}_{i=1}^p.$$

Let $\mathbf{r}^+ = (r_1^+, \ldots, r_p^+)^\top$ and $\mathbf{r}^- = (r_1^-, \ldots, r_p^-)^\top$. For simplicity, let $\mathcal{A}_1(\mathbf{Y}) := \mathcal{A}(\bar{\mathbf{X}}\mathbf{Y}^\top)$. Then, minimizing $F(\bar{\mathbf{X}}, \mathbf{Y})$ given $\bar{\mathbf{X}}$ in Step 5 is equivalent to the following QP:

$$\begin{aligned} \min_{\mathbf{Y}\in\mathbb{R}^{n\times k}, \mathbf{r}^+, \mathbf{r}^- \in \mathbb{R}_+^p} \quad & \omega\|\mathbf{r}^+\|_2^2 + (1-\omega)\|\mathbf{r}^-\|_2^2 \\ \text{s.t.} \quad & \mathbf{r}^+ - \mathbf{r}^- = \mathbf{b} - \mathcal{A}_1(\mathbf{Y}). \end{aligned} \tag{5.3}$$

Similarly, Step 7 can be reformulated as a QP as well.

Steps 5 and 7 can be solved even more efficiently in the matrix completion case, which aims at recovering an incomplete low-rank matrix from a few observed entries and is a special case of the matrix estimation problem under discussion, where each b_i is simply an observation of a matrix element (possibly with noise). In matrix completion, we can decompose the above QP in Steps 5 and 7 by updating each row of $\mathbf{X}$ (or $\mathbf{Y}$), whose time complexity in practice is similar to conventional alternating least squares, e.g., [42], which also solve QPs.

5.4 Theoretical Results

We now show that the proposed algorithm for expectile matrix factorization retains the optimality for any $\omega \in (0, 1)$ when observations are noiseless, i.e., the produced $\mathbf{M}^{(T)}$ will eventually approach the true low-rank matrix $\mathbf{M}^*$ to be recovered. We generalize the recent result [57] of the optimality of alternating minimization for matrix estimation under the symmetric least squares loss function (corresponding to $\omega = 0.5$ in EMF) to a general class of "asymmetric least squares" loss functions with any $\omega \in (0, 1)$.

We assume that the linear mapping $\mathcal{A}$ satisfies the well-known $2k$-RIP condition [108]:

Assumption 5.1 ($2k$-RIP)**.** *There exists a constant $\delta_{2k} \in (0, 1)$ such that for any matrix M with rank at most $2k$, the following property holds:*

$$(1 - \delta_{2k})\|\mathbf{M}\|_F^2 \le \|\mathcal{A}(\mathbf{M})\|_2^2 \le (1 + \delta_{2k})\|\mathbf{M}\|_F^2.$$

A linear mapping $\mathcal{A}$ satisfying the RIP condition can be obtained in various ways. For example, if each entry of $\mathbf{A}_i$ is independently drawn from the sub-

Gaussian distribution, then $\mathcal{A}$ satisfies $2k$-RIP property with high probability for $p = \Omega(\delta_{2k}^{-2} kn \log n)$ [108].

Clearly, Algorithm 5 involves minimizing a weighted sum of squared losses in the form of

$$\mathcal{F}(\mathbf{X}, \mathbf{Y}) = \sum_{i=1}^{p} w_i (b_i - \langle \mathbf{A}_i, \mathbf{X}\mathbf{Y}^\top \rangle)^2,$$

although the weight w_i depends on the sign of residual r_i and may vary in each iteration. We show that if the weights w_i are confined within a closed interval $[w_-, w_+]$ with constants $w_-, w_+ > 0$, then the alternating minimization algorithm for the weighted sum of squared losses will converge to an optimal point. Without loss of generality, we can assume that $w_- \leq 1/2 \leq w_+$ and $w_- + w_+ = 1$ by weight normalization.

The first step is to prove strongly convexity and smoothness of $\mathcal{F}(\mathbf{X}, \mathbf{Y})$ if one variable is fixed by a orthonormal matrix as follows:

Lemma 5.4.1. *Suppose that δ_{2k} and $\bar{\mathbf{X}}^{(t)}$ satisfy*

$$\delta_{2k} \leq \frac{\sqrt{2} w_-^2 (1 - \delta_{2k})^2 \sigma_k}{24 \xi w_+ k (1 + \delta_{2k}) \sigma_1}. \tag{5.4}$$

and

$$\|\bar{\mathbf{X}}^{(t)} - \bar{\mathbf{U}}^{(t)}\|_F \leq \frac{w_- (1 - \delta_{2k}) \sigma_k}{2 \xi w_+ (1 + \delta_{2k}) \sigma_1} \tag{5.5}$$

Then we have:

$$\|\mathbf{Y}^{(t+0.5)} - \mathbf{V}^{(t)}\|_F \leq \frac{\sigma_k}{2\xi} \|\bar{\mathbf{X}}^{(t)} - \bar{\mathbf{U}}^{(t)}\|_F.$$

Clearly, Algorithm 5 involves minimizing a weighted sum of squared losses in the form of $\mathcal{F}(\mathbf{X}, \mathbf{Y}) = \sum_{i=1}^{p} w_i (b_i - \langle \mathbf{A}_i, \mathbf{X}\mathbf{Y}^\top \rangle)^2$, although the weight w_i depends on the sign of residual r_i and may vary in each iteration. We show that the if the weights w_i are confined in a closed interval $[w_-, w_+]$ with constants $w_-, w_+ > 0$, then the alternating minimization algorithm for the weighted sum of squared losses will converge to the optimal point. Without loss of generality, we can assume that $w_- \leq 1/2 \leq w_+$ and $w_- + w_+ = 1$ by weight normalization. For notation simplicity, we denote a finite positive constant $\xi > 1$ throughout this thesis.

Lemma 5.4.2. *Suppose the linear operator $\mathcal{A}(\cdot)$ satisfies $2k$-RIP with parameter δ_{2k}. For any orthonormal matrix $\mathbf{s}\bar{X} \in \mathbb{R}^{m \times k}$, the function $\mathcal{F}(\bar{\mathbf{X}}, \mathbf{Y})$ with bounded*

weights is strongly convex and smooth. In particular, if any weight w_i in $\mathcal{F}(\bar{\mathbf{X}}, \mathbf{Y})$ belongs to $[w_-, w_+]$, the value of

$$\mathcal{F}(\bar{\mathbf{X}}, \mathbf{Y}') - \mathcal{F}(\bar{\mathbf{X}}, \mathbf{Y}) - \langle \nabla_{\mathbf{Y}} \mathcal{F}(\bar{\mathbf{X}}, \mathbf{Y}), \mathbf{Y}' - \mathbf{Y} \rangle$$

is bounded by

$$[w_-(1-\delta_{2k})\|\mathbf{Y}' - \mathbf{Y}\|_F^2, w_+(1+\delta_{2k})\|\mathbf{Y}' - \mathbf{Y}\|_F^2]$$

for all $\mathbf{Y}, \mathbf{Y}'$.

Lemma 5.4.2 shows that $\mathcal{F}(\mathbf{X}, \mathbf{Y})$ can be block-wise strongly convex and smooth if the weights w_i belongs to $[w_-, w_+]$. In the following, we use $\mathbf{U}$ and $\mathbf{V}$ to denote the optimal factorization of $\mathbf{M}^* = \mathbf{U}\mathbf{V}^\top$. Note that $\mathbf{U}$ and $\mathbf{V}$ are unique up to orthogonal transformations. The following lemma shows that by taking the block-wise minimum, the distance between the newly updated variable $\mathbf{Y}^{(t+0.5)}$ and its "nearby" $\mathbf{V}^{(t)}$ is upper bounded by the distance between $\mathbf{X}^{(t)}$ and its corresponding neighbor $\mathbf{U}^{(t)}$.

Lemma 5.4.3. *Suppose that δ_{2k} satisfies*

$$\delta_{2k} \leq \frac{w_-^2(1-\delta_{2k})^2\sigma_k^4}{48\xi^2 k w_+^2(1+\delta_{2k})^2\sigma_1^4}.$$

We have $\|\bar{\mathbf{Y}}^{(t+1)} - \bar{\mathbf{V}}^{(t+1)}\|_F \leq \frac{1}{\xi}\|\bar{\mathbf{X}}^{(t)} - \bar{\mathbf{U}}^{(t)}\|_F$.

The following lemma shows an upper bound after initialization:

Lemma 5.4.4. *Suppose that δ_{2k} satisfies*

$$\delta_{2k} \leq \frac{w_-^2(1-\delta_{2k})^2\sigma_k^4}{48\xi^2 k w_+^2(1+\delta_{2k})^2\sigma_1^4}.$$

Then there exists a factorization of $M^* = \bar{\mathbf{U}}^0\mathbf{V}^{(0)\top}$ such that $\bar{\mathbf{U}}^{(0)} \in \mathbb{R}^{m\times k}$ is an orthonormal matrix, and satisfies

$$\|\bar{\mathbf{X}}^{(0)} - \bar{\mathbf{U}}^{(0)}\|_F \leq \frac{w_-(1-\delta_{2k})\sigma_k}{2\xi w_+(1+\delta_{2k})\sigma_1}.$$

With the above three lemmas, we can now provide our convergence analysis by iteratively upper bounded the distance $\|\bar{\mathbf{Y}}^{(t)} - \bar{\mathbf{V}}^{(t)}\|_F$ as well as $\|\bar{\mathbf{X}}^{(t)} - \bar{\mathbf{U}}^{(t)}\|_F$. First, we show the geometric convergence of alternating minimization for weighted matrix factorization, if all weights belong to $[w_-, w_+]$ in each iteration:

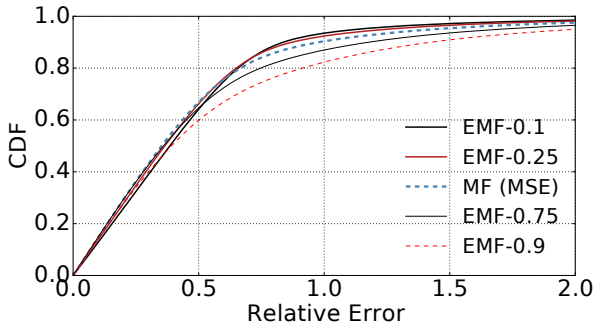
Theorem 5.4.1. *Assume that the linear mapping $\mathcal{A}(\cdot)$ satisfies $2k$-RIP condition with $\delta_{2k} \leq C_1/k \cdot w_-^2/w_+^2$ for some small constant C_1, and assume that the singular values of $\mathbf{M}^*$ are bounded in the range of $[\sigma_{\min}(\mathbf{M}^*), \sigma_{\max}(\mathbf{M}^*)]$, where singular values in $\mathbf{M}^*$ are constants and do not scale with the matrix size. Suppose the weights in $\mathcal{F}(\mathbf{X}, \mathbf{Y})$ are bounded by two positive finite constants, i.e., $w_i \in [w_-, w_+]$ with $0 < w_- \leq 1/2 \leq w_+ < 1$ and $w_- + w_+ = 1$. Then, given any desired precision ε, there exists a constant C_2 such that by applying alternating minimization to $\mathcal{F}(\mathbf{X}, \mathbf{Y})$, the solution $\mathbf{M}^{(T)}$ satisfies $\|\mathbf{M}^{(T)} - \mathbf{M}^*\|_F \leq \varepsilon$ for all $T \geq O(\log(C_2/\varepsilon) + \log(w_-/w_+))$.*

The proof of this theorem is in Sec. 5.6. Theorem 5.4.1 implies that the weighted matrix factorization can geometrically converge to a global optimum. Note that the negative term $\log(w_-/w_+)$ does not imply that weighted matrix factorization converges faster, since the value of C_2 for two w's may differ. In fact, due to the lower RIP constant δ_{2k}, the convergence rate in the case of $w_- \neq w_+$ is usually slower than that in the case of $w_- = w_+$.

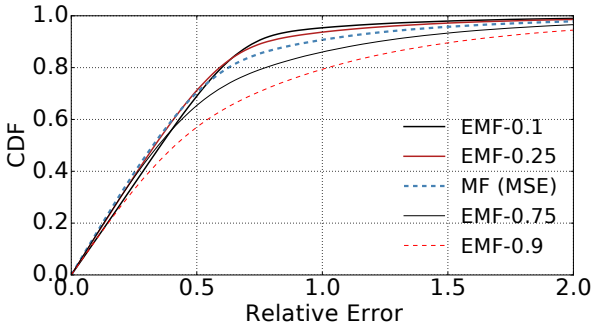
In Algorithm 5 for expectile matrix factorization, the weight w_i in each iteration for residual r_i is ω if $r_i \geq 0$, and is $1 - \omega$ otherwise. Although w_i is changing across iterations, we can choose $w_- = \min(\omega, 1-\omega)$ and $w_+ = \max(\omega, 1-\omega)$, both satisfying the assumptions in Theorem 5.4.1, to bound all w_i. Then we can derive the following main result directly from Theorem 5.4.1.

Theorem 5.4.2 (Optimality of Algorithm 5)**.** *Suppose $\omega \leq 1/2$. Assume that the linear mapping $\mathcal{A}(\cdot)$ satisfies $2k$-RIP condition with $\delta_{2k} \leq C_3/k \cdot (1-\omega)^2/\omega^2$ for some small constant C_3, and assume that the singular values of $\mathbf{M}^*$ are bounded in the range of $[\sigma_{\min}(\mathbf{M}^*), \sigma_{\max}(\mathbf{M}^*)]$, where singular values in $\mathbf{M}^*$ are constants and do not scale with the matrix size. Then, given any desired precision ε, there exists a constant C_4 such that Algorithm 5 satisfies $\|\mathbf{M}^{(T)} - \mathbf{M}^*\|_F \leq \varepsilon$ for all $T \geq O(\log(C_4/\varepsilon) + \log(\omega/(1-\omega)))$. If $\omega > 1/2$, we can get the same result by substituting ω with $1 - \omega$.*

Additionally, we can further determine the sampling complexity, the number of observations needed for exact recovery, as $p = \Omega\big(\frac{(1-\omega)^2}{\omega^2} k^3 n \log n\big)$, if the entries of A_i are independently drawn from a sub-Gaussian distribution with zero mean and unit variance, since we require $\delta_{2k} \leq C/k \cdot (1-\omega)^2/\omega^2$. This also matches the sampling complexity of conventional matrix factorization [108].



(a) Sampling rate $R = 0.05$



(b) Sampling rate $R = 0.1$

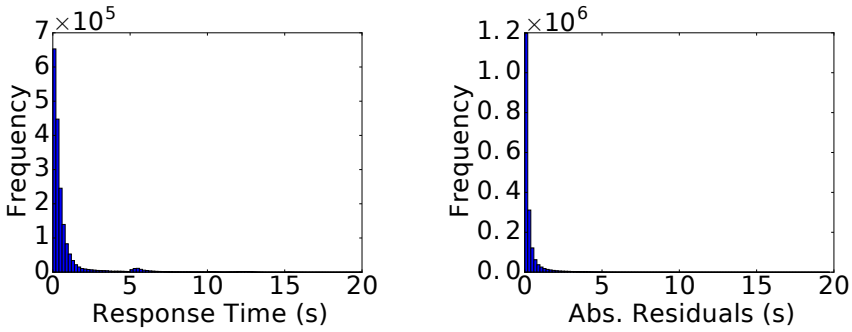
Figure 5.2: CDF of relative errors via expectile matrix factorization on synthetic 1000×1000 matrices with skewed noise.

5.5 Experiments

In this section, we evaluate the performance of EMF in comparison to the state-of-the-art MSE-based matrix factorization based on both skewed synthetic data and a real-world dataset containing web service response times between 339 users and 5825 web services collected worldwide [50]. In both tasks, we aim to estimate a true matrix M^* based on partial observations. We define the relative error (RE) as $|\mathbf{M}^*_{i,j} - \hat{\mathbf{M}}_{i,j}|/\mathbf{M}^*_{i,j}$ for all the missing entries (i, j). We use RE to evaluate the prediction accuracy of different methods under a certain sampling rate R (the fraction of known entries).

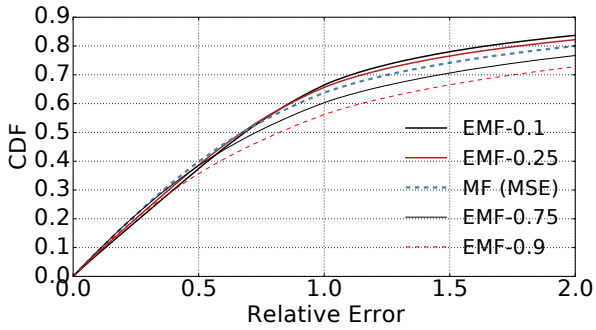
5.5.1 Experiments on Skewed Synthetic Data

We randomly generate a 1000×1000 matrix $\mathbf{M}^* = \mathbf{X}\mathbf{Y}^\top$ of rank $k = 10$, where $\mathbf{X} \in \mathbb{R}^{m \times k}$ and $\mathbf{Y} \in \mathbb{R}^{n \times k}$ have independent and uniformly distributed entries in $[0, 1]$. Then, we contaminate $\mathbf{M}^*$ by a skewed noise matrix $0.5\mathbf{N}$, where $\mathbf{N}$ contains independent *Chi-square* entries with 3 degrees of freedom. The 0.5 is to make

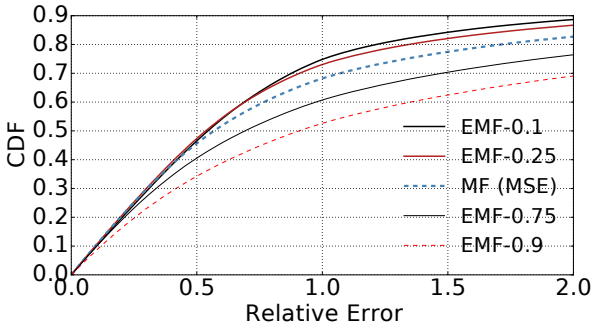


(a) Response times (b) Residuals from MSE-MF

Figure 5.3: Histograms of a) response times between 5825 web services and 339 service users; b) the residuals of estimates from MSE-based matrix factorization applied on the complete matrix.



(a) Sampling rate $R = 0.05$

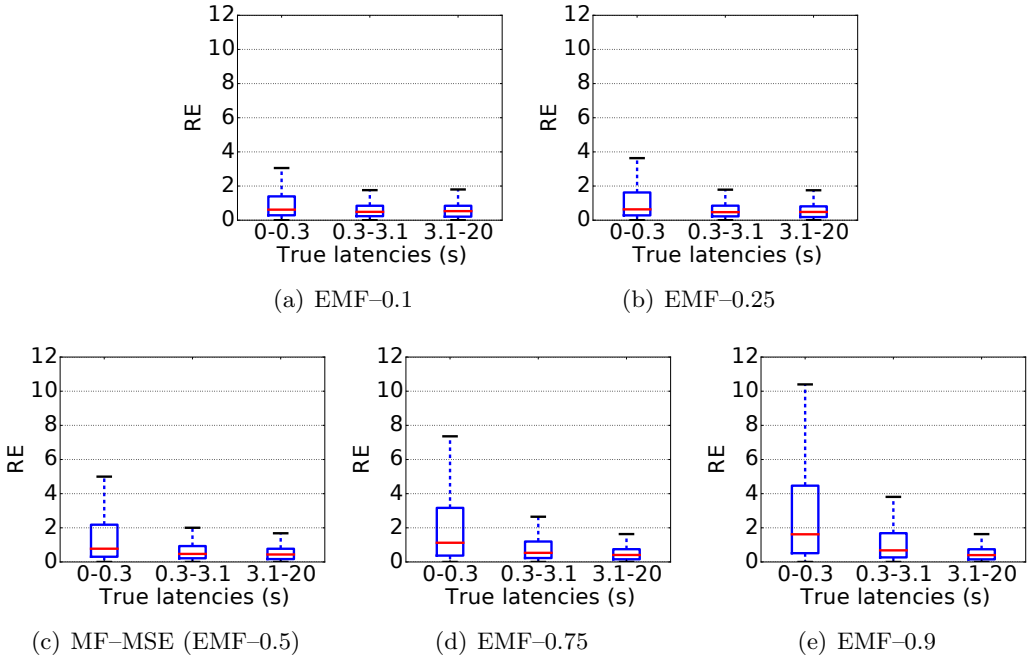


(b) Sampling rate $R = 0.1$

Figure 5.4: CDF of relative errors via expectile matrix factorization for web service response time estimation under different sampling rates and ω.

sure the noise does not dominate. We observe some elements in the contaminated matrix and aim to recover the underlying true low-rank $\mathbf{M}^*$ under two sampling rates $R = 0.05$ and $R = 0.1$, respectively, where R is the fraction of elements observed. The experiment is repeated for 10 times for each R. We plot the CDF of relative errors in terms of recovering the missing elements of $\mathbf{M}^*$ in Fig. 5.2. We can see that

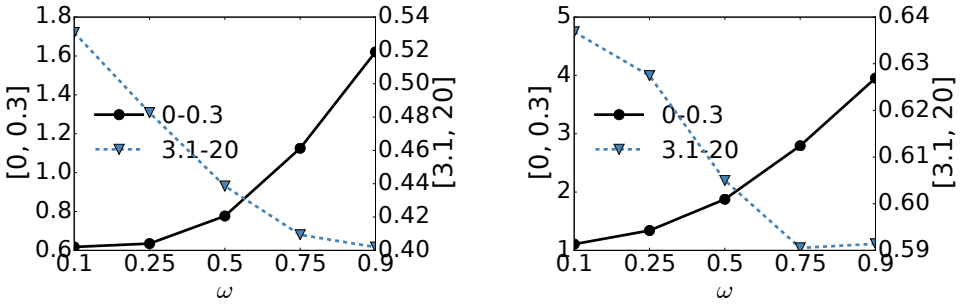
expectile matrix factorization outperforms the conventional MSE-based algorithm (EMF with $\omega = 0.5$) in terms of recovery from skewed noise, with $\omega = 0.1$ yielding the best performance, under both $R = 0.05$ and $R = 0.1$. When more observations are available with $R = 0.1$, EMF with $\omega = 0.1$ demonstrates more benefit as it is more robust to the heavy-tailed noise in data.



(a) EMF–0.1 (b) EMF–0.25

(c) MF–MSE (EMF–0.5) (d) EMF–0.75 (e) EMF–0.9

Figure 5.5: Box plots of relative errors for different bins of true latencies in the test sets.



(a) Median of REs (b) IQR of REs

Figure 5.6: The medians and IQRs of relative errors for different bins as ω varies.

5.5.2 Experiments on Web Service Latency Estimation

In these experiments, we aim to recover the web service response times between 339 users and 5825 web services [50] distributed worldwide, under different sampling rates.

Fig. 5.3(a) shows the histogram of all the response times measured between 339 users and 5825 web services. While most entries are less than 1 second, some response times may be as high as 20 seconds due to network delay variations, software glitches and even temporary service outages. The mean latency is 0.91 second, whereas the median is only 0.32 second. This implies that the mean is heavily impacted by the few tail values, while the 0.1-th expectile, which is 0.3 second, is closer to the median of the data. Therefore, if we use the conventional MSE-based matrix factorization to recover this skewed data, the result can be far away from the central area, while EMF with $\omega = 0.1$ may better explain the central tendency.

We further performed the MSE-based matrix factorization for the complete response time matrix, which boils down to singular value decomposition (SVD) and we plot the residual histogram in Fig. 5.3(b). In this figure, 90% of residuals are less than 0.8, while the largest residual can be up to 19.73. Since the residuals are still highly skewed, the conditional means do not serve as good estimates for the most probable data.

In Fig. 5.4, we plot the relative errors of recovering missing response times with EMF under different ω. Note that EMF-0.5 is essentially the conventional MSE-based matrix factorization. In Fig. 5.4, we can see that EMF-0.1 performs the best under both sampling rates, and EMF-0.9 performs the worst, because the 0.1-th expectile is the closest to the median, while both the mean and 0.9-th expectile are far away from the central area of data distribution.

To take a closer look at the performance EMF on different segments of data, we divide the testing response times into three bins: 0-0.3s containing 47.5% of all entries, 0.3-3.1s containing 45.4% of all entries, and 3.1-20s containing only 7.1% of all entries. We show the relative errors for testing samples from different bins in box plots in Fig. 5.5 under different ω. In addition, in Fig. 5.6, we plot the median of REs and the interquartile range (IQR, the gap between the upper and lower quartiles) of REs when $R = 0.1$, as ω varies for the lower latency bin and the higher latency bin, respectively.

We can observe that EMF with a lower ω achieves higher accuracy in the lower range 0-0.3s, while EMF with a higher ω can predict better in the higher end 3.1-20s. This observation conforms to the intuition illustrated in Fig. 5.1: an $\omega < 0.5$ penalizes negative residuals, pushing the estimates to be more accurate on the lower end, where most data are centered around. From Fig. 5.6(a) and Fig. 5.6(b), we can see that EMF-0.1 predicts the best for the lower range, while EMF-0.9 performs the best for the higher range. However, since most data are distributed in the lower range, EMF-0.1 is better at predicting the central tendency and achieves the best overall accuracy.

5.6 Detailed Proofs for Theoretical Results

5.6.1 Preliminaries

Lemma 5.6.1 (Lemma B.1 of [108]). *Suppose $\mathcal{A}(\cdot)$ satisfies $2k$-RIP. For any $\mathbf{X}, \mathbf{U} \in \mathbb{R}^{m\times k}$ and $\mathbf{Y}, \mathbf{V} \in \mathbb{R}^{n\times k}$, we have*

$$|\langle \mathcal{A}(\mathbf{X}\mathbf{Y}^\top), \mathcal{A}(\mathbf{U}\mathbf{V}^\top)\rangle - \langle \mathbf{X}^\top\mathbf{U}, \mathbf{Y}^\top\mathbf{V}\rangle| \leq 3\delta_{2k}\|\mathbf{X}\mathbf{Y}^\top\|_F \cdot \|\mathbf{U}\mathbf{V}^\top\|_F$$

Lemma 5.6.2 (Lemma 2.1 of [109]). *Let $\mathbf{b} = \mathcal{A}(\mathbf{M}^*) + \boldsymbol{\varepsilon}$, where $\mathbf{M}^*$ is a matrix with the rank of k, $\mathcal{A}$ is the linear mapping operator satisfies $2k$-RIP with constant $\delta_{2k} < 1/3$, and $\boldsymbol{\varepsilon}$ is a bounded error vector. Let $\mathbf{M}^{(t+1)}$ be the $t+1$-th step iteration of SVP, then we have*

$$\|\mathcal{A}(\mathbf{M}^{(t+1)}) - \mathbf{b}\|_2^2 \leq \|\mathcal{A}(\mathbf{M}^*) - \mathbf{b}\|_2^2 + 2\delta_{2k}\|\mathcal{A}(\mathbf{M}^{(t)}) - \mathbf{b}\|_2^2.$$

Lemma 5.6.3 (Lemma 4.5 of [57]). *Suppose that $\mathbf{Y}^{(t+0.5)}$ in Alg. 5 satisfies the following:*

$$\|\mathbf{Y}^{(t+0.5)} - \mathbf{V}^{(t)}\|_F \leq \sigma_k/4.$$

Then, there exists a factorization of matrix $\mathbf{M}^* = \mathbf{U}^{(t+1)}\bar{\mathbf{V}}^{(t+1)\top}$ such that $\mathbf{V}^{(t+1)} \in \mathbb{R}^{n\times k}$ is an orthonormal matrix, and satisfies

$$\|\bar{\mathbf{Y}}^{(t+1)} - \bar{\mathbf{V}}^{(t+1)}\|_F \leq 2/\sigma_k \cdot \|\mathbf{Y}^{(t+0.5)} - \mathbf{V}^{(t)}\|_F.$$

5.6.2 Proof of Lemma 5.4.2

Now we begin to prove these lemmas. Note that a similar technique has also been used by [57]. Since we should fix $\mathbf{X}^{(t)}$ or $\mathbf{Y}^{(t)}$ as orthonormal matrices, we perform a QR decomposition after getting the minimum. The following lemma shows the distance between $\bar{\mathbf{Y}}^{(t+1)}$ and its "nearby" $\bar{\mathbf{V}}^{(t+1)}$ is still under control. Due to the page limit, we leave all the proofs in the supplemental material.

Proof. Since $\mathcal{F}(\bar{\mathbf{X}}, \mathbf{Y})$ is a quadratic function, we have

$$\begin{aligned}\mathcal{F}(\bar{\mathbf{X}}, \mathbf{Y}') &= \mathcal{F}(\bar{\mathbf{X}}, \mathbf{Y}) + \langle \nabla_Y \mathcal{F}(\bar{\mathbf{X}}, \mathbf{Y}), \mathbf{Y}' - \mathbf{Y} \rangle \\ &\quad + \frac{1}{2}(\mathrm{vec}(\mathbf{Y}') - \mathrm{vec}(\mathbf{Y}))^\top \nabla_Y^2 \mathcal{F}(\bar{\mathbf{X}}, \mathbf{Y})(\mathrm{vec}(\mathbf{Y}') - \mathrm{vec}(\mathbf{Y})),\end{aligned}$$

and it suffices to bound the singular values of the Hessian matrix $\mathbf{S}_\omega := \nabla_{\mathbf{Y}}^2 \mathcal{F}(\bar{\mathbf{X}}, \mathbf{Y})$ so that

$$\begin{aligned}\mathcal{F}(\bar{\mathbf{X}}, \mathbf{Y}') - \mathcal{F}(\bar{\mathbf{X}}, \mathbf{Y}) - \langle \nabla_Y \mathcal{F}(\bar{\mathbf{X}}, \mathbf{Y}), \mathbf{Y}' - \mathbf{Y} \rangle &\leq \frac{\sigma_{\max}(S_\omega)}{2} \|\mathbf{Y}' - \mathbf{Y}\|_F^2 \\ \mathcal{F}(\bar{\mathbf{X}}, \mathbf{Y}') - \mathcal{F}(\bar{\mathbf{X}}, \mathbf{Y}) - \langle \nabla_Y \mathcal{F}(\bar{\mathbf{X}}, \mathbf{Y}), \mathbf{Y}' - \mathbf{Y} \rangle &\geq \frac{\sigma_{\min}(S_\omega)}{2} \|\mathbf{Y}' - \mathbf{Y}\|_F^2.\end{aligned}$$

Now we proceed to derive the Hessian matrix $\mathbf{S}_\omega$. Using the fact $\mathrm{vec}(\mathbf{AXB}) = (\mathbf{B}^\top \otimes \mathbf{A})\mathrm{vec}(\mathbf{X})$, we can write $\mathbf{S}_\omega$ as follows:

$$\begin{aligned}\mathbf{S}_\omega &= \sum_{i=1}^{p} 2w_i \cdot \mathrm{vec}(\mathbf{A}_i^\top \bar{\mathbf{X}}) \mathrm{vec}^\top (\mathbf{A}_i^\top \bar{\mathbf{X}}) \\ &= \sum_{i=1}^{p} 2w_i \cdot (\mathbf{I}_k \otimes \mathbf{A}_i^\top) \mathrm{vec}(\bar{\mathbf{X}}) \mathrm{vec}^\top (\bar{\mathbf{X}}) (\mathbf{I}_k \otimes \mathbf{A}_i).\end{aligned}$$

Consider a matrix $\mathbf{Z} \in \mathbb{R}^{n \times k}$ with $\|\mathbf{Z}\|_F = 1$, and we denote $\mathbf{z} = \mathrm{vec}(\mathbf{Z})$. Then we have

$$\begin{aligned}\mathbf{z}^\top \mathbf{S}_\omega \mathbf{z} &= \sum_{i=1}^{p} 2w_i \cdot \mathbf{z}^\top (\mathbf{I}_k \otimes \mathbf{A}_i^\top) \mathrm{vec}(\bar{\mathbf{X}}) \mathrm{vec}^\top (\bar{\mathbf{X}}) (\mathbf{I}_k \otimes \mathbf{A}_i) \mathbf{Z} \\ &= \sum_{i=1}^{p} 2w_i \cdot \mathrm{vec}^\top (\mathbf{A}_i \mathbf{Z}) \mathrm{vec}(\bar{\mathbf{X}}) \mathrm{vec}^\top (\bar{\mathbf{X}}) \mathrm{vec}(\mathbf{A}_i \mathbf{Z}) \\ &= \sum_{i=1}^{p} 2w_i \cdot \mathrm{tr}^2 (\bar{\mathbf{X}}^\top \mathbf{A}_i \mathbf{Z}) = \sum_{i=1}^{p} 2w_i \cdot \mathrm{tr}^2 (\mathbf{A}_i^\top \bar{\mathbf{X}} \mathbf{Z}^\top).\end{aligned}$$

From the $2k$-RIP property of $\mathbf{A}_\rangle$, we have

$$\begin{aligned}\mathbf{z}^\top \mathbf{S}_\omega \mathbf{z} &\leq \sum_{i=1}^{p} 2w_+ \mathrm{tr}^2 (\bar{\mathbf{X}}^\top \mathbf{A}_i \mathbf{Z}) \\ &\leq 2w_+(1 + \delta_{2k}) \|\bar{\mathbf{X}} \mathbf{Z}^\top\|_F \\ &= 2w_+(1 + \delta_{2k}) \|\mathbf{Z}^\top\|_F = 2w_+(1 + \delta_{2k}).\end{aligned}$$

Similarly, we also have

$$\mathbf{z}^\top \mathbf{S}_\omega \mathbf{z} \geq 2w_-(1-\delta_{2k}).$$

Therefore, the maximum singular value $\sigma_{\max}$ is upper bounded by $2w_+(1+\delta_{2k})$ and the minimum singular value $\sigma_{\min}$ is lower bounded by $2w_-(1-\delta_{2k})$, and the Lemma has been proved. □

5.6.3 Proof of Lemma 5.4.1

We prove this lemma by introducing a divergence function as follows.

$$\begin{aligned}
&\mathcal{D}(\mathbf{Y}^{(t+0.5)}, \mathbf{Y}^{(t+0.5)}, \bar{\mathbf{X}}^{(t)}) \\
&= \left\langle \nabla_Y \mathcal{F}(\bar{\mathbf{U}}^{(t)}, \mathbf{Y}^{(t+0.5)}) - \nabla_Y \mathcal{F}(\bar{\mathbf{X}}^{(t)}, \mathbf{Y}^{(t+0.5)}), \frac{\mathbf{Y}^{(t+0.5)} - \mathbf{V}^{(t)}}{\|\mathbf{Y}^{(t+0.5)} - \mathbf{V}^{(t)}\|_F} \right\rangle.
\end{aligned}$$

The following lemma helps us bound $\mathcal{D}(\mathbf{Y}^{(t+0.5)}, \mathbf{Y}^{(t+0.5)}, \bar{\mathbf{X}}^{(t)})$.

Lemma 5.6.4. *Under the same condition in Lemma 5.4.1, we have*

$$\mathcal{D}(\mathbf{Y}^{(t+0.5)}, \mathbf{Y}^{(t+0.5)}, \bar{\mathbf{X}}^{(t)}) \leq \frac{3(1-\delta_{2k})\sigma_k}{2\xi} \cdot \frac{w_+^2}{w_-} \|\bar{\mathbf{X}}^{(t)} - \bar{\mathbf{U}}^{(t)}\|. \tag{5.6}$$

Lemma 5.6.5.

$$\|\mathbf{Y}^{(t+0.5)} - \mathbf{V}^{(t)}\|_F \leq \frac{1}{2w_-(1-\delta_{2k})} \mathcal{D}(\mathbf{Y}^{(t+0.5)}, \mathbf{Y}^{(t+0.5)}, \bar{\mathbf{X}}^{(t)}). \tag{5.7}$$

Given Lemma 5.6.4 and Lemma 5.6.5, we can now bound $\|\mathbf{Y}^{(t+0.5)} - \mathbf{V}^{(t)}\|_F$ and thus prove Lemma 5.4.1.

Proof of Lemma 5.4.1. From Lemma 5.6.4, we have

$$\mathcal{D}(\mathbf{Y}^{(t+0.5)}, \mathbf{Y}^{(t+0.5)}, \bar{\mathbf{X}}^{(t)}) \leq \frac{(1-\delta_{2k})\sigma_k w_-}{\xi} \|\bar{\mathbf{X}}^{(t)} - \bar{\mathbf{U}}^{(t)}\|_F,$$

and from Lemma 5.6.5, we have

$$\|\mathbf{Y}^{(t+0.5)} - \mathbf{V}^{(t)}\|_F \leq \frac{1}{2w_-(1-\delta_{2k})} \mathcal{D}(\mathbf{Y}^{(t+0.5)}, \mathbf{Y}^{(t+0.5)}, \hat{X}^{(t)}).$$

Therefore,

$$\begin{aligned}
& \|\mathbf{Y}^{(t+0.5)} - \mathbf{V}^{(t)}\|_F && (5.8)\\
\leq\ & \frac{(1-\delta_{2k})\sigma_k w_-}{\xi} \cdot \frac{1}{2w_-(1-\delta_{2k})} \|\bar{\mathbf{X}}^{(t)} - \bar{\mathbf{U}}^{(t)}\|_F && (5.9)\\
=\ & \frac{\sigma_k}{2\xi} \|\bar{\mathbf{X}}^{(t)} - \bar{\mathbf{U}}^{(t)}\|_F && (5.10)
\end{aligned}$$

□

5.6.4 Proof of Lemma 5.4.3

From Lemma 5.4.1, we have

$$\begin{aligned} \|\mathbf{Y}^{(0.5)} - \mathbf{V}^{(t)}\|_F &\leq \frac{\sigma_k}{2\xi}\|\bar{\mathbf{X}}^{(t)} - \bar{\mathbf{U}}_F^{(t)}\| && (5.11) \\ &\leq \frac{(1-\delta_{2k})\sigma_k w_-}{2\xi^2(1+\delta_{2k})\sigma_1 w_+} \leq \frac{\sigma_k}{4}, && (5.12) \end{aligned}$$

where (5.12) is from $\xi > 1$. Thus, we can see from Lemma5.6.3 and we obtain that

$$\|\bar{\mathbf{Y}}^{(t+1)} - \bar{\mathbf{V}}^{(t+1)}\|_F \leq \frac{2}{\sigma_k}\|\mathbf{Y}^{(0.5)} - \mathbf{V}^{(t)}\|_F \leq \frac{1}{\xi}\|\bar{\mathbf{X}}^{(t)} - \bar{\mathbf{U}}^{(t)}\| \leq \frac{(1-\delta_{2k})\sigma_k w_-}{2\xi(1+\delta_{2k})\sigma_1 w_+}. \quad (5.13)$$

5.6.5 Proof of Lemma 5.4.4

Proof of Lemma 5.4.4. The initialization step can be regarded as taking a step iterate of singular value projection (SVP) as taking $\mathbf{M}^{(t)} = 0$ and the next iterate with the step size $1/(1+\delta_{2k})$ will result $\mathbf{M}^{(t+1)} = \bar{\mathbf{X}}^{(0)}\mathbf{D}^{(0)}\bar{\mathbf{Y}}^{(0)}/(1+\delta_{2k})$, where $\bar{\mathbf{X}}^{(0)}$, $\mathbf{D}^{(0)}$ and $\bar{\mathbf{Y}}^{(0)}$ are from the top k singular value decomposition of $\sum_{i=1}^p b_i\mathbf{A}_i$.

Then, by Lemma 5.6.2 and the fact that $\boldsymbol{\varepsilon} = 0$, we have

$$\left\|\mathcal{A}\left(\frac{\bar{\mathbf{X}}^{(0)}\mathbf{D}^{(0)}\bar{\mathbf{Y}}^{(0)}}{1+\delta_{2k}}\right) - \mathcal{A}(\mathbf{M}^*)\right\|_2^2 \leq 4\delta_{2k}\|0 - \mathcal{A}(\mathbf{M}^*)\|_2^2. \quad (5.14)$$

From the $2k$-RIP condition, we have

$$\begin{aligned} \left\|\frac{\bar{\mathbf{X}}^{(0)}\mathbf{D}^{(0)}\bar{\mathbf{Y}}^{(0)}}{1+\delta_{2k}}\right\| &\leq \frac{1}{1-\delta_{2k}}\left\|\mathcal{A}\left(\frac{\bar{\mathbf{X}}^{(0)}\mathbf{D}^{(0)}\bar{\mathbf{Y}}^{(0)}}{1+\delta_{2k}}\right) - \mathcal{A}(\mathbf{M}^*)\right\|_2^2 \\ &\leq \frac{4\delta_{2k}}{1-\delta_{2k}}\|\mathcal{A}(\mathbf{M}^*)\|_2^2 \\ &\leq \frac{4\delta_{2k}(1+\delta_{2k})}{1-\delta_{2k}}\|\mathbf{M}^*\|_F^2 \leq 6\delta_{2k}\|\mathbf{M}^*\|_F^2. \end{aligned}$$

Then, we project each column of $\mathbf{M}^*$ into the column subspace of $\bar{\mathbf{X}}^{(0)}$ and obtain

$$\|(\bar{\mathbf{X}}^{(0)}\bar{\mathbf{X}}^{(0)\top} - \mathbf{I})\mathbf{M}^*\|_F^2 \leq 6\delta_{2k}\|\mathbf{M}^*\|_F^2.$$

We denote the orthonormal complement of $\bar{\mathbf{X}}^{(0)}$ as $\bar{\mathbf{X}}_\perp^{(0)}$. Then, we have

$$\frac{6\delta_{2k}k\sigma_1^2}{\sigma_k^2} \geq \|\bar{\mathbf{X}}_\perp^{(0)\top}\bar{\mathbf{U}}^*\|_F^2,$$

where $\bar{\mathbf{U}}^*$ is from the singular value decomposition of $\mathbf{M}^* = \bar{\mathbf{U}}\mathbf{D}\bar{\mathbf{V}}^\top$. Then, there exists a unitary matrix $\mathbf{O} \in \mathbb{R}^{k\times k}$ such that $\mathbf{O}^\top\mathbf{O} = \mathbf{I}_k$ and

$$\|\bar{\mathbf{X}}^{(0)} - \bar{\mathbf{U}}^*\mathbf{O}\|_F \leq \sqrt{2}\|\bar{\mathbf{X}}_\perp^{(0)\top}\bar{\mathbf{U}}^*\|_F \leq 2\sqrt{3\delta_{2k}\frac{\sigma_1}{\sigma_k}}.$$

By taking the condition of δ_{2k}, we have

$$\|\bar{\mathbf{X}}^0 - \bar{\mathbf{U}}^*\|_F \leq \frac{(1-\delta_{2k})\sigma_k w_-}{2\xi(1+\delta_{2k})\sigma_1 w_+}. \tag{5.15}$$

□

5.6.6 Proof of Theorem 5.4.1

Proof of Theorem 5.4.1. The proof of Theorem 5.4.1 can be done by induction. Firstly, we note that Lemma 5.4.4 ensures that the initial $\bar{\mathbf{X}}^{(0)}$ is close to a $\bar{\mathbf{U}}^{(0)}$. Then, by Lemma 5.6.3 we have the following sequence of inequalities for all T iterations:

$$\begin{aligned} &\|\bar{\mathbf{Y}}^{(T)} - \bar{\mathbf{V}}^{(T)}\|_F \\ &\leq \frac{1}{\xi}\|\bar{\mathbf{X}}^{(T-1)} - \bar{\mathbf{U}}^{(T-1)}\|_F \\ &\leq \cdots \leq \frac{1}{\xi^{2T-1}}\|\bar{\mathbf{X}}^{(0)} - \bar{\mathbf{U}}^{(0)}\|_F \\ &\leq \frac{(1-\delta_{2k})\sigma_k w_-}{2\xi^{2T}(1+\delta_{2k})\sigma_1 w_+}. \end{aligned} \tag{5.16}$$

Therefore, we can bound the right most term by $\varepsilon/2$ for any given precision ε. By algebra, we can derive the required number of iterations T as:

$$T \geq \frac{1}{2}\log\left(\frac{(1-\delta_{2k})\sigma_k w_-}{2\varepsilon(1+\delta_{2k})\sigma_1 w_+}\right)\log^{-1}\xi.$$

Similarly, we can also bound $\|\mathbf{X}^{(T-0.5)} - \mathbf{U}^{(T)}\|_F$,

$$\|\mathbf{X}^{(T-0.5)} - \mathbf{U}^{(T)}\|_F \leq \frac{\sigma_k}{2\xi}\|\bar{\mathbf{Y}}^{(T)} - \bar{\mathbf{V}}^{(T)}\|_F \leq \frac{(1-\delta_{2k})\sigma_k^2 w_-}{4\xi(1+\delta_{2k})\sigma_1 w_+}. \tag{5.17}$$

To make it smaller than $\varepsilon\sigma_1/2$, we need the number of iterations as

$$T \geq \frac{1}{2}\log\left(\frac{(1-\delta_{2k})\sigma_k^2 w_-}{4\varepsilon(1+\delta_{2k})\sigma_1 w_+}\right)\log^{-1}\xi.$$

Combining all results we have

$$\begin{aligned} & \|\mathbf{M}^{(T)} - \mathbf{M}^*\|_F & (5.18)\\ = \;& \|\mathbf{X}^{(T-0.5)}\bar{\mathbf{Y}}^{(T)\top} - \mathbf{U}^{(T)}\bar{\mathbf{V}}^{(T)\top}\|_F \\ = \;& \|\mathbf{X}^{(T-0.5)}\bar{\mathbf{Y}}^{(T)\top} - \mathbf{U}^{(T)}\bar{\mathbf{Y}}^{(T)\top} + \mathbf{U}^{(T)}\bar{\mathbf{Y}}^{(T)\top} - \mathbf{U}^{(T)}\bar{\mathbf{V}}^{(T)\top}\|_F \\ \leq \;& \|\bar{\mathbf{Y}}^{(T)\top}\|_2\|\mathbf{X}^{(T-0.5)} - \mathbf{U}^{(T)}\|_F + \|\mathbf{U}^{(T)}\|_2\|\bar{\mathbf{Y}}^{(T)} - \bar{\mathbf{V}}^{(T)}\|_F \leq \varepsilon. & (5.19) \end{aligned}$$

Here we use the fact that the orthonormal matrix $\bar{\mathbf{V}}^{(T)}$ leads to $\|\bar{\mathbf{V}}^{(T)}\|_2 = 1$, and $\|\mathbf{M}^*\|_2 = \|\mathbf{U}^{(T)}\bar{\mathbf{V}}^{(T)\top}\|_2 = \|\mathbf{U}^{(T)}\|_2 = \sigma_1$. Now we complete the proof of Theorem 5.4.1. □

5.6.7 Proofs for auxiliary lemmas

Proof of Lemma 5.6.4. In this proof we omit the iteration superscript, and $\mathbf{Y}$ stands particularly for $\mathbf{Y}^{(t+0.5)}$. Since b_i is measured by $\langle \mathbf{A}_i, \bar{\mathbf{U}}\mathbf{V}^\top \rangle$, we have

$$\mathcal{F}(\bar{\mathbf{X}}, \mathbf{Y}) = \sum_{i=1}^{p} w_i (\langle \mathbf{A}_i, \bar{\mathbf{X}}\mathbf{Y}^\top \rangle - \langle \mathbf{A}_i, \bar{\mathbf{U}}\mathbf{V}^\top \rangle)^2.$$

By taking the partial derivatives on $\mathbf{Y}$ we have

$$\begin{aligned} \nabla_{\mathbf{Y}} \mathcal{F}(\bar{\mathbf{X}}, \mathbf{Y}) &= \sum_{i=1}^{p} 2w_i (\langle \mathbf{A}_i, \bar{\mathbf{X}}\mathbf{Y}^\top \rangle - \langle \mathbf{A}_i, \bar{\mathbf{U}}\mathbf{V}^\top \rangle) \mathbf{A}_i^\top \mathbf{X} \\ &= \sum_{i=1}^{p} 2w_i (\langle \mathbf{A}_i^\top \bar{\mathbf{X}}, \mathbf{Y} \rangle - \langle \mathbf{A}_i^\top \bar{\mathbf{U}}, \mathbf{V} \rangle) \mathbf{A}_i^\top \mathbf{X} \end{aligned}$$

Let $\mathbf{x} := \mathrm{vec}(\bar{\mathbf{X}})$, $\mathbf{y} := \mathrm{vec}(\mathbf{Y})$, $\mathbf{u} := \mathrm{vec}(\bar{\mathbf{U}})$, and $\mathbf{v} := \mathrm{vec}(\mathbf{V})$. Since $\mathbf{Y}$ minimizes $\mathcal{F}(\bar{\mathbf{X}}, \hat{\mathbf{Y}})$, we have

$$\begin{aligned} & \mathrm{vec}(\nabla_{\mathbf{Y}} \mathcal{F}(\bar{\mathbf{X}}, \mathbf{Y})) \\ = & \sum_{i=1}^{p} 2w_i (\langle \mathbf{A}_i^\top \bar{\mathbf{X}}, \mathbf{Y} \rangle - \langle \mathbf{A}_i^\top \bar{\mathbf{U}}, \mathbf{V} \rangle) \mathbf{A}_i^\top \bar{\mathbf{X}} \\ = & \sum_{i=1}^{p} 2w_i (\mathrm{vec}(\mathbf{A}_i^\top \bar{\mathbf{X}}) \cdot \langle \mathbf{A}_i^\top \bar{\mathbf{X}}, \mathbf{Y} \rangle - \mathrm{vec}(\mathbf{A}_i^\top \bar{\mathbf{X}}) \cdot \langle \mathbf{A}_i^\top \bar{\mathbf{X}}, \mathbf{Y} \rangle) \\ = & \sum_{i=1}^{p} 2w_i ((\mathbf{I}_k \otimes \mathbf{A}_i^\top) \mathbf{x}\mathbf{x}^\top (I_k \otimes \mathbf{A}_i) \mathbf{y} - (\mathbf{I}_k \otimes \mathbf{A}_i^\top) \mathbf{x}\mathbf{u}^\top (\mathbf{I}_k \otimes \mathbf{A}_i) \mathbf{v}) \end{aligned}$$

We denote

$$\mathbf{S}_\omega = \sum_{i=1}^{p} 2w_i \cdot (\mathbf{I}_k \otimes \mathbf{A}_i^\top) \mathbf{x}\mathbf{x}^\top (\mathbf{I}_k \otimes \mathbf{A}_i),$$

and

$$\mathbf{J}_\omega = \sum_{i=1}^{p} 2w_i \cdot (\mathbf{I}_k \otimes \mathbf{A}_i^\top) \mathbf{x}\mathbf{u}^\top (\mathbf{I}_k \otimes \mathbf{A}_i),$$

So the equation becomes $\mathbf{S}_\omega \mathbf{y} - \mathbf{J}_\omega \mathbf{v} = 0$ and since $\mathbf{S}_\omega$ is invertible we have $\mathbf{y} = (\mathbf{S}_\omega)^{-1} \mathbf{J}_\omega \mathbf{v}$. Meanwhile, we denote

$$\mathbf{G}_\omega = \sum_{i=1}^{p} 2w_i \cdot (\mathbf{I}_k \otimes \mathbf{A}_i^\top) \mathbf{u}\mathbf{u}^\top (\mathbf{I}_k \otimes \mathbf{A}_i)$$

as the Hessian matrix of $\nabla^2_{\mathbf{Y}}\mathcal{F}(\bar{\mathbf{U}},\mathbf{Y})$. Then, the partial gradient $\nabla_{\mathbf{Y}}\mathcal{F}(\bar{\mathbf{U}},\mathbf{Y})$ can be written as

$$\begin{aligned}
\mathrm{vec}(\nabla_{\mathbf{Y}}\mathcal{F}(\bar{\mathbf{U}},\mathbf{Y})) &= \sum_{i=1}^{p} 2w_i\left(\langle \mathbf{A}_i^\top\bar{\mathbf{U}},\mathbf{Y}\rangle - \langle \mathbf{A}_i^\top\bar{\mathbf{U}},\mathbf{V}\rangle\right)\left(\mathbf{I}_k\otimes\mathbf{A}_i^\top\right)\mathbf{u}\\
&= \sum_{i=1}^{p} 2w_i\left((\mathbf{I}_k\otimes\mathbf{A}_i^\top)\mathbf{u}\mathbf{u}^\top(\mathbf{I}_k\otimes\mathbf{A}_i)\mathbf{y} - (\mathbf{I}_k\otimes\mathbf{A}_i^\top)\mathbf{u}\mathbf{u}^\top(\mathbf{I}_k\otimes\mathbf{A}_i)\mathbf{v}\right)\\
&= \mathbf{G}_\omega(\mathbf{y}-\mathbf{v})\\
&= \mathbf{G}_\omega(\mathbf{S}_\omega^{-1}\mathbf{J}_\omega - \mathbf{I}_{nk})\mathbf{v}.
\end{aligned}$$

Since we have $\mathrm{vec}(\nabla_{\mathbf{Y}}\mathcal{F}(\bar{\mathbf{X}},\mathbf{Y}))=0$, the divergence becomes

$$\mathcal{D} = \langle \nabla_{\mathbf{Y}}(\bar{\mathbf{U}},\mathbf{Y}), (\mathbf{Y}-\mathbf{V})/\|\mathbf{Y}-\mathbf{V}\|_F\rangle.$$

So we need to bound $\nabla_{\mathbf{Y}}\mathcal{F}(\bar{\mathbf{U}},\mathbf{Y})$. Let $\mathbf{K} := \bar{\mathbf{X}}^\top\bar{\mathbf{U}}\otimes\mathbf{I}_n$. To get the estimate of $\mathbf{S}_\omega^{-1}\mathbf{J}_\omega - \mathbf{I}_{nk}$, we rewrite it as

$$\mathbf{S}_\omega^{-1}\mathbf{J}_\omega - \mathbf{I}_{nk} = \mathbf{K} - \mathbf{I}_{nk} + \mathbf{S}_\omega^{-1}(\mathbf{J}_\omega - \mathbf{S}_\omega\mathbf{K}).$$

We firstly bound the term $(\mathbf{K}-\mathbf{I}_{nk})\mathbf{v}$. Recall $\mathrm{vec}(\mathbf{AXB}) = (\mathbf{B}^\top\otimes\mathbf{A})\mathrm{vec}(\mathbf{X})$, we have

$$\begin{aligned}
(\mathbf{K}-\mathbf{I}_{nk})\mathbf{v} &= \left((\bar{\mathbf{X}}^\top\bar{\mathbf{U}}-\mathbf{I}_k)\otimes\mathbf{I}_n\right)\mathbf{v} = \mathrm{vec}\left(\mathbf{V}(\bar{\mathbf{U}}^\top X - \mathbf{I}_k)\right)\\
\|(\mathbf{K}-\mathbf{I}_{nk})\mathbf{v}\|_2 &= \|\mathbf{V}(\bar{\mathbf{U}}^\top\bar{\mathbf{X}}-\mathbf{I}_k)\|_F \le \sigma_1\|\bar{\mathbf{U}}^\top\bar{\mathbf{X}}-\mathbf{I}_k\|_F\\
&\le \sigma_1\|(\bar{\mathbf{X}}-\bar{\mathbf{U}})^\top(\bar{\mathbf{X}}-\bar{\mathbf{U}})\|_F \le \sigma_1\|\bar{\mathbf{X}}-\bar{\mathbf{U}}\|_F^2
\end{aligned}$$

We then bound the term $\mathbf{J}_\omega - \mathbf{S}_\omega\mathbf{K}$. For any two matrices $\mathbf{Z}_1,\mathbf{Z}_2\in\mathbb{R}^{n\times k}$, we denote $\mathbf{z}_1 := \mathrm{vec}(\mathbf{Z}_1)$ and $\mathbf{z}_2 := \mathrm{vec}(\mathbf{Z}_2)$. Then we have:

$$\begin{aligned}
&\mathbf{z}_1^\top(\mathbf{S}_\omega\mathbf{K}-\mathbf{J}_\omega)\mathbf{z}_2\\
=\ & \sum_{i=1}^{p} 2w_i\mathbf{z}_1^\top\left((\mathbf{I}_k\otimes\mathbf{A}_i^\top)\mathbf{x}\mathbf{x}^\top(\mathbf{I}_k\otimes\mathbf{A}_i)(\bar{\mathbf{X}}^\top\bar{\mathbf{U}}\otimes\mathbf{I}_n) - \mathbf{U}^\top(\mathbf{I}_k\otimes\mathbf{A}_i)\right)\mathbf{z}_2\\
=\ & \sum_{i=1}^{p} 2w_i\langle\mathbf{Z}_1,\mathbf{A}_i^\top\bar{\mathbf{X}}\rangle\cdot\left(\mathbf{X}^\top(\bar{\mathbf{X}}^\top\bar{\mathbf{U}}\otimes\mathbf{A}_i)\mathbf{z}_2 - \langle\bar{\mathbf{U}},\mathbf{A}_i\mathbf{Z}\rangle\right)\\
=\ & \sum_{i=1}^{p} 2w_i\langle\mathbf{A}_i,\bar{\mathbf{X}}\mathbf{Z}_1^\top\rangle\langle\mathbf{A}_i,(\bar{\mathbf{X}}\bar{\mathbf{X}}^\top-\mathbf{I}_m)\bar{\mathbf{U}}\mathbf{Z}_2^\top\rangle\\
\le\ & 2w_+\langle\mathcal{A}(\bar{\mathbf{X}}\mathbf{Z}_1^\top),\mathcal{A}((\bar{\mathbf{X}}\bar{\mathbf{X}}^\top-\mathbf{I}_m)\bar{\mathbf{U}}\mathbf{Z}_2^\top)\rangle
\end{aligned}$$

Since $\bar{\mathbf{X}}^\top(\bar{\mathbf{X}}\bar{\mathbf{X}}^\top - \mathbf{I}_m)\bar{\mathbf{U}} = 0$, by Lemma 5.6.1 we have

$$\begin{aligned}
& \mathbf{z}_1^\top(\mathbf{S}_\omega\mathbf{K} - \mathbf{J}_\omega)\mathbf{z}_2 \\
\leq\ & 2w_+ \cdot 3\delta_{2k}\|\bar{\mathbf{X}}\mathbf{Z}_1^\top\|_F\|(\bar{\mathbf{X}}\bar{\mathbf{X}}^\top - \mathbf{I}_m)\bar{\mathbf{U}}\mathbf{Z}_2^\top\|_F \\
\leq\ & 6w_+\delta_{2k}\|\mathbf{Z}_1\|_F\sqrt{\|\bar{\mathbf{U}}^\top(\bar{\mathbf{X}}\bar{\mathbf{X}}^\top - \mathbf{I}_m)\bar{\mathbf{U}}\|_F\|\mathbf{Z}_2^\top\mathbf{Z}_2\|_F} \\
=\ & 6w_+\delta_{2k}\sqrt{\|\bar{\mathbf{U}}^\top(\bar{\mathbf{X}}\bar{\mathbf{X}}^\top - \mathbf{I}_m)\bar{\mathbf{U}}\|_F} \\
\leq\ & 6w_+\delta_{2k}\sqrt{2k}\|\bar{\mathbf{X}} - \bar{\mathbf{U}}\|_F.
\end{aligned}$$

Thus, the spectral norm of this term is upper bounded by $6w_+\delta_{2k}\sqrt{2k}\|\bar{\mathbf{X}} - \bar{\mathbf{U}}\|_F$ and finally we have

$$\begin{aligned}
& \|\mathrm{vec}(\nabla_Y\mathcal{F}(\bar{\mathbf{U}}, \mathbf{Y}))\|_2 \\
=\ & \|\mathbf{G}_\omega(\mathbf{S}_\omega^{-1}\mathbf{J}_\omega - \mathbf{I}_{nk})\mathbf{v}\|_2 \\
\leq\ & w_+(1+\delta_{2k})\left(\sigma_1\|\bar{\mathbf{X}} - \bar{\mathbf{U}}\|_F^2 + \frac{1}{(1-\delta_{2k})w_-}\|\mathbf{S}_\omega\mathbf{K} - \mathbf{J}_\omega\|_2\|\mathbf{V}\|_F\right) \\
\leq\ & w_+(1+\delta_{2k})\left(\sigma_1\|\bar{\mathbf{X}} - \bar{\mathbf{U}}\|_F^2 + \frac{\sigma_1\sqrt{k}}{(1-\delta_{2k})w_-}\|\mathbf{S}_\omega\mathbf{K} - \mathbf{J}_\omega\|_2\right) \\
\leq\ & w_+(1+\delta_{2k})\sigma_1\left(\|\bar{\mathbf{X}} - \bar{\mathbf{U}}\|_F^2 + \frac{\sqrt{k}\cdot 6w_+\delta_{2k}\sqrt{2k}}{(1-\delta_{2k})w_-}\|\bar{\mathbf{X}} - \bar{\mathbf{U}}\|_F\right) \\
\leq\ & w_+(1+\delta_{2k})\sigma_1\left(\|\bar{\mathbf{X}} - \bar{\mathbf{U}}\|_F^2 + \frac{6\sqrt{2}\cdot w_+\delta_{2k}k}{(1-\delta_{2k})w_-}\|\bar{\mathbf{X}} - \bar{\mathbf{U}}\|_F\right).
\end{aligned}$$

Under the given condition, we can upper bound $\|\bar{\mathbf{X}} - \bar{\mathbf{U}}\|$ and δ_{2k} and we go to the final step as follows:

$$\begin{aligned}
\|\mathrm{vec}(\nabla_{\mathbf{Y}}\mathcal{F}(\bar{\mathbf{U}}, \mathbf{Y}))\|_2 &\leq \frac{(1-\delta_{2k})\sigma_k w_-}{2\xi} + \frac{(1-\delta_{2k})\sigma_k w_-}{2\xi} \\
&= \frac{(1-\delta_{2k})\sigma_k w_-}{\xi}
\end{aligned}$$

Thus, the divergence $\mathcal{D}(\mathbf{Y}, \mathbf{Y}, \bar{\mathbf{X}})$ can be upperbounded by

$$\mathcal{D}(\mathbf{Y}, \mathbf{Y}, \bar{\mathbf{X}}) \leq \|\mathrm{vec}(\nabla_{\mathbf{Y}}\mathcal{F}(\bar{\mathbf{U}}, \mathbf{Y}))\|_2 \leq \frac{(1-\delta_{2k})\sigma_k w_-}{\xi}\|\bar{\mathbf{X}}^{(t)} - \bar{\mathbf{U}}^{(t)}\|_F. \tag{5.20}$$

□

Proof of Lemma 5.6.5. Here we utilize the strongly convexity of $\mathcal{F}(\mathbf{X}, \mathbf{Y})$ given a orthonormal matrix $\mathbf{X}$. By Lemma 5.4.2, we have

$$\mathcal{F}(\bar{\mathbf{U}}, \mathbf{V}) \geq \mathcal{F}(\bar{\mathbf{U}}, \mathbf{Y}) + \langle\nabla_{\mathbf{Y}}\mathcal{F}(\bar{\mathbf{U}}, \mathbf{Y}), \mathbf{V} - \mathbf{Y}\rangle + w_-(1-\delta_{2k})\|\mathbf{V} - \mathbf{Y}\|_F^2. \tag{5.21}$$

Since $\mathbf{V}$ minimizes the function $\mathcal{F}(\bar{\mathbf{U}}, \hat{\mathbf{V}})$, we have $\langle \nabla_{\mathbf{Y}} \mathcal{F}(\bar{\mathbf{U}}, \mathbf{V}), \mathbf{Y} - \mathbf{V} \rangle \geq 0$ and thus

$$\begin{aligned} \mathcal{F}(\bar{\mathbf{U}}, \mathbf{Y}) &\geq \mathcal{F}(\bar{\mathbf{U}}, \mathbf{V}) + \langle \nabla_{\mathbf{Y}} \mathcal{F}(\bar{\mathbf{U}}, \mathbf{V}), \mathbf{Y} - \mathbf{V} \rangle + (1 - \delta_{2k}) w_- \|\mathbf{V} - \mathbf{Y}\|_F^2 \\ &\geq \mathcal{F}(\bar{\mathbf{U}}, \mathbf{V}) + w_-(1 - \delta_{2k}) \|\mathbf{V} - \mathbf{Y}\|_F^2. \end{aligned} \tag{5.22}$$

Add (5.21) and (5.22) we have

$$\langle \nabla_{\mathbf{Y}} \mathcal{F}(\bar{\mathbf{U}}, \mathbf{Y}), \mathbf{Y} - \mathbf{V} \rangle \geq 2w_-(1 - \delta_{2k}) \|\mathbf{V} - \mathbf{Y}\|_F^2. \tag{5.23}$$

Since $\mathbf{Y}$ also minimizes $\mathcal{F}(\bar{\mathbf{X}}, \hat{\mathbf{Y}})$, we have $\langle \nabla_{\mathbf{Y}} \mathcal{F}(\bar{\mathbf{X}}, \mathbf{V}), \mathbf{V} - \mathbf{Y} \rangle \geq 0$ and thus

$$\begin{aligned} \langle \nabla_{\mathbf{Y}} \mathcal{F}(\bar{\mathbf{U}}, \mathbf{Y}) - \nabla_{\mathbf{Y}} \mathcal{F}(\bar{\mathbf{X}}, \mathbf{Y}), \mathbf{Y} - \mathbf{V} \rangle &\geq \langle \nabla_{\mathbf{Y}} \mathcal{F}(\bar{\mathbf{U}}, \mathbf{Y}), \mathbf{Y} - \mathbf{V} \rangle \\ &\geq 2w_-(1 - \delta_{2k}) \|\mathbf{V} - \mathbf{Y}\|_F^2. \end{aligned} \tag{5.24}$$

Therefore, we have

$$\|\mathbf{V} - \mathbf{Y}\|_F \leq \frac{1}{2w_-(1 - \delta_{2k})} \mathcal{D}(\mathbf{Y}, \mathbf{Y}, \bar{\mathbf{X}}) \tag{5.25}$$

□

Chapter 6

Asynchronous Blockwise ADMM

6.1 Background

The need to scale up machine learning in the presence of sheer volume of data has spurred recent interest in developing efficient distributed optimization algorithms. In the rest of this thesis, we will study the following form of optimization problems:

$$\begin{aligned} \min_{\mathbf{x}} \quad & f(x_1, \ldots, x_M) + \sum_{j=1}^{M} h_j(x_j), \\ \text{s.t.} \quad & x_j \in \mathcal{X}_j, j = 1, \ldots, M \end{aligned} \tag{6.1}$$

where $f : \mathcal{X} \to \mathbb{R}$ is a smooth but possibly *nonconvex* function that fitting the model $\mathbf{x} := (x_1, \ldots, x_M)$ from a dataset $\mathcal{D}$ that has n samples. When the dataset $\mathcal{D}$ is large, we can either split $\mathcal{D}$ into N parts and distribute them to N machines, or store it in a third-party distributed storage system like HDFS, Amazon S3 and Google Cloud Storage. In this chapter, we study distributed optimization algorithms in the former fashion, and we discuss optimization algorithms of the latter setting in Chapter 7 and Chapter 8.

Suppose we have N machines, so now the problem in (6.1) becomes:

$$\begin{aligned} \min_{\mathbf{x}} \quad & \sum_{i=1}^{N} f_i(x_1, \ldots, x_M) + \sum_{j=1}^{M} h_j(x_j), \\ \text{s.t.} \quad & x_j \in \mathcal{X}_j, j = 1, \ldots, M \end{aligned} \tag{6.2}$$

where each $f_i : \mathcal{X} \to \mathbb{R}$ fits the model $\mathbf{x} := (x_1, \ldots, x_M)$ to local training data available on node i; each $\mathcal{X}_j$ is a closed, convex, and compact set; and the regularizer

$h(\mathbf{x}) := \sum_{j=1}^{M} h_j(x_j)$ is a separable, convex but possibly *nonsmooth* regularization term to prevent overfitting.

To date, a number of efficient asynchronous and distributed stochastic gradient descent (SGD) algorithms, e.g., [32, 75, 16], have been proposed, in which each worker node asynchronously updates its local model or gradients based on its local dataset, and sends them to the server(s) for model updates or aggregation. Yet, SGD is not particularly suitable for solving optimization problems with nonsmooth objectives or with constraints, which are prevalent in practical machine learning adopting regularization, e.g., [12]. Distributed (synchronous) ADMM [2, 77, 79, 80, 78, 81, 82, 110] has been widely studied as an alternative method, which avoids the common pitfalls of SGD for highly nonconvex problems, such as saturation effects, poor conditioning, and saddle points [110]. The original idea on distributed ADMM can be found in [2], which is essentially a synchronous algorithm. In this work, we focus on studying the asynchronous distributed alternating direction method of multipliers (ADMM) for nonconvex nonsmooth optimization.

Asynchronous distributed ADMM has been actively discussed in recent literature. Zhang and Kwok [77] consider an asynchronous ADMM assuming bounded delay, which enables each worker node to update a local copy of the model parameters asynchronously without waiting for other workers to complete their work, while a single server is responsible for driving the local copies of model parameters to approach the global *consensus variables*. They provide proof of convergence for convex objective functions only. Wei and Ozdaglar [81] assume that communication links between nodes can fail randomly, and propose an ADMM scheme that converges almost surely to a saddle point. Chang *et al.* [79, 80] propose an asynchronous ADMM algorithm with analysis for nonconvex objective functions. However, their work requires each worker to solve a subproblem *exactly*, which is often costly in practice. Hong [78] proposes another asynchronous ADMM algorithm, where each worker only computes the gradients based on local data, while all model parameter updates happen at a single server, a possible bottleneck in large clusters.

To our knowledge, all existing work on asynchronous distributed ADMM requires locking global consensus variables at the (single) server for each model update; although asynchrony is allowed among workers, i.e., workers are allowed to be at different iterations of model updating. Such atomic or memory-locking operations essentially serialize model updates contributed by different workers, which may se-

riously limit the algorithm scalability. In many practical problems, not all workers need to access all model parameters. For example, in recommender systems, a local dataset of user-item interactions is only associated with a specific set of users (and items), and therefore does not need to access the latent variables of other users (or items). In text categorization, each document usually consists of a subset of words or terms in corpus, and each worker only needs to deal with the words in its own local corpus.

It is worth noting that enabling *block-wise* updates in ADMM is critical for training large models, such as sparse logistic regression, robust matrix completion, etc., since not all worker nodes will need to work on all model parameters — each worker only needs to work on the blocks of parameters pertaining to its local dataset. For these reasons, block-wise updates have been extensively studied for a number of gradient type of distributed optimization algorithms, including SGD [75], proximal gradient descent [84], block or stochastic coordinate descent (BCD or SCD) [73], as well as for a recently proposed block successive upper bound minimization method (BSUM) [103].

In this work, we propose the first *block-wise* asynchronous distributed ADMM algorithm that can increase efficiency over existing single-server ADMM algorithms, by better exploiting the parallelization opportunity in model parameter updates. Specifically, we introduce the *general form consensus optimization* problem [2], and solve it in a *block-wise* asynchronous fashion, thus making ADMM amenable for implementation on Parameter Server, with multiple servers hosting model parameters. In our algorithm, each worker only needs to work on one or multiple blocks of parameters that are relevant to its local data, while different blocks of model parameters can be updated in parallel asynchronously subject to a bounded delay. Since this scheme does not require locking all the decision variables together, it belongs to the set of *lock-free* optimization algorithms (e.g., HOGWILD! [32] as a lock-free version of SGD) in the literature. Our scheme is also useful on shared memory systems, such as on a single machine with multi-cores or multiple GPUs, where enforcing atomicity on all the consensus variables is inefficient. Theoretically, we prove that, for general *nonconvex* objective functions, our scheme can converge to stationary points. The results in this chapter have appeared in [5].

6.2 Preliminaries

6.2.1 Consensus Optimization and ADMM

The minimization in (6.2) can be reformulated into a *global variable consensus optimization* problem [2]:

$$\min_{\mathbf{z},\{\mathbf{x}_i\}\in\mathcal{X}} \quad \sum_{i=1}^{N} f_i(\mathbf{x}_i) + h(\mathbf{z}), \tag{6.3a}$$

$$\text{s.t.} \quad \mathbf{x}_i = \mathbf{z}, \quad \forall i = 1, \ldots, N, \tag{6.3b}$$

where $\mathbf{z}$ is often called the *global consensus variable*, traditionally stored on a master node, and $\mathbf{x}_i$ is its local copy updated and stored on one of N worker nodes. The function h is decomposable. It has been shown [2] that such a problem can be efficiently solved using distributed (synchronous) ADMM. In particular, let $\mathbf{y}_i$ denote the Lagrange dual variable associated with each constraint in (6.3b) and define the Augmented Lagrangian as

$$\begin{aligned} L(\mathbf{X}, \mathbf{Y}, \mathbf{z}) = \sum_{i=1}^{N} f_i(\mathbf{x}_i) + h(\mathbf{z}) + \sum_{i=1}^{N} \langle \mathbf{y}_i, \mathbf{x}_i - \mathbf{z} \rangle \\ + \sum_{i=1}^{N} \frac{\rho_i}{2} \|\mathbf{x}_i - \mathbf{z}\|^2, \end{aligned} \tag{6.4}$$

where $\mathbf{X} := (\mathbf{x}_1, \ldots, \mathbf{x}_N)$ represents a juxtaposed matrix of all $\mathbf{x}_i$, and $\mathbf{Y}$ represents the juxtaposed matrix of all $\mathbf{y}_i$. We have, for (synchronized) rounds $t = 0, 1, \ldots$, the following variable updating equations:

$$\begin{aligned} \mathbf{x}_i^{t+1} &= \arg\min_{\mathbf{x}_i \in \mathcal{X}} f_i(\mathbf{x}_i) + \langle \mathbf{y}_i^t, \mathbf{x}_i - \mathbf{z}^t \rangle + \frac{\rho_i}{2} \|\mathbf{x}_i - \mathbf{z}^t\|^2, \\ \mathbf{y}_i^{t+1} &= \mathbf{y}_i^t + \rho_i (\mathbf{x}_i^{t+1} - \mathbf{z}^t), \\ \mathbf{z}^{t+1} &= \arg\min_{\mathbf{z} \in \mathcal{X}} h(\mathbf{z}) + \sum_{i=1}^{N} \langle \mathbf{y}_i^t, \mathbf{x}_i^t - \mathbf{z}^t \rangle + \sum_{i=1}^{N} \frac{\rho_i}{2} \|\mathbf{x}_i^t - \mathbf{z}^t\|^2. \end{aligned}$$

6.2.2 General Form Consensus Optimization

Many machine learning problems involve highly sparse models, in the sense that each local dataset on a worker is only associated with a few model parameters, i.e., each f_i only depends on a subset of the elements in $\mathbf{x}$. The global consensus optimization problem in (6.3), however, ignores such sparsity, since in each round each worker i must push the entire vectors $\mathbf{x}_i$ and $\mathbf{y}_i$ to the master node to update $\mathbf{z}$. In fact, this is the setting of all recent work on asynchronous distributed ADMM,

e.g., [77]. In this case, when multiple workers attempt to update the global census variable $\mathbf{z}$ at the same time, $\mathbf{z}$ must be locked to ensure atomic updates, which leads to diminishing efficiency as the number of workers N increases.

To better exploit model sparsity in practice for further parallelization opportunities between workers, we consider the *general form consensus optimization* problem [2]. Specifically, with N worker nodes and M server nodes, the vectors $\mathbf{x}_i$, $\mathbf{y}_i$ and $\mathbf{z}$ can all be decomposed into M blocks. Let z_j denote the j-th block of the global consensus variable $\mathbf{z}$, located on server j, for $j = 1, \ldots, M$. Similarly, let $x_{i,j}$ ($y_{i,j}$) denote the corresponding j-th block of the local variable $\mathbf{x}_i$ ($\mathbf{y}_i$) on worker i. Let $\mathcal{E}$ be all the (i, j) pairs such that f_i depends on the block $x_{i,j}$ (and correspondingly depends on z_j). Furthermore, let $\mathcal{N}(j) = \{i|(i, j) \in \mathcal{E}\}$ denote the set of all the neighboring workers of server j. Similarly, let $\mathcal{N}(i) = \{j|(i, j) \in \mathcal{E}\}$.

Then, the *general form consensus problem* [2] is described as follows:

$$\begin{aligned} \min_{\mathbf{z},\mathbf{X}} \quad & \sum_{i=1}^{N} f_i(\mathbf{x}_i) + h(\mathbf{z}), \\ \text{s.t.} \quad & x_{i,j} = z_j, \quad \forall (i, j) \in \mathcal{E}, \\ & x_{i,j}, z_j \in \mathcal{X}_j. \end{aligned} \tag{6.5}$$

The structure of problem (6.5) can effectively capture the sparsity inherent to many practical machine learning problems. Since each f_i only depends on a few blocks, the formulation in (6.5) essentially reduces the number of decision variables—it does not matter what value $x_{i,j}$ will take for any $(i, j) \notin \mathcal{E}$. For example, when training a topic model for documents, the feature of each document is represented as a bag of words, and hence only a subset of all words in the vocabulary will be active in each document's feature. In this case, the constraint $x_{i,j} = z_j$ only accounts for those words j that appear in the document i, and therefore only those words j that appeared in document i should be optimized. Like (6.4), we also define the Augmented Lagrangian $L(\mathbf{X}, \mathbf{Y}, \mathbf{z})$ as follows:

$$\begin{aligned} L(\mathbf{X}, \mathbf{Y}, \mathbf{z}) = \sum_{i=1}^{N} f_i(\mathbf{x}_i) + h(\mathbf{z}) + \sum_{(i,j)\in\mathcal{E}} \langle y_{i,j}, x_{i,j} - z_j \rangle \\ + \sum_{(i,j)\in\mathcal{E}} \frac{\rho_i}{2} \|x_{i,j} - z_j\|^2. \end{aligned}$$

The formulation in (6.5) perfectly aligns with the latest *Parameter Server* architecture (see Fig. 1.1). Here we can let each server node maintain one model block

z_j, such that worker i updates z_j if and only if $(i,j) \in \mathcal{E}$. Since all three vectors $\mathbf{x}_i$, $\mathbf{y}_i$ and $\mathbf{z}$ in (6.5) are decomposable into blocks, to achieve a higher efficiency, we will investigate block-wise algorithms which not only enable different workers to send their updates asynchronously to the server (like prior work on asynchronous ADMM does), but also enable different model blocks $\mathbf{z}_j$ to be updated in parallel and asynchronously on different servers, removing the locking or atomicity assumption required for updating the entire $\mathbf{z}$.

6.3 A Block-wise, Asynchronous, and Distributed ADMM Algorithm

In this section, we present our proposed *block-wise*, *asynchronous* and *distributed* ADMM algorithm (a.k.a, AsyBADMM) for the general consensus problem. For ease of presentation, we first describe a synchronous version motivated by the basic distributed ADMM for *nonconvex* optimization problems as a starting point.

6.3.1 Block-wise Synchronous ADMM

The update rules presented in Sec. 6.2.1 represent the basic synchronous distributed ADMM approach [2]. To solve the general form consensus problem, our block-wise version extends such a synchronous algorithm mainly by 1) approximating the update rule of $\mathbf{x}_i$ with a simpler expression under nonconvex objective functions, and 2) converting the all-vector updates of variables into block-wise updates only for $(i,j) \in \mathcal{E}$.

Generally speaking, in each synchronized iteration t, each worker node i updates all blocks of its local primal variables $x_{i,j}$ and dual variables $y_{i,j}$ for $j \in \mathcal{N}(i)$, and pushes these updates to the corresponding servers. Each server j, when it has received $x_{i,j}$ and $y_{i,j}$ from all $i \in \mathcal{N}(j)$, will update z_j accordingly, by aggregating these received blocks.

Specifically, at iteration t, the basic synchronous distributed ADMM will do the following update for $\mathbf{x}_i$:

$$\mathbf{x}_i^{t+1} = \arg\min_{\mathbf{x}_i} f_i(\mathbf{x}_i) + \sum_{j \in \mathcal{N}(i)} \langle y_{i,j}^t, x_{i,j} - {z_j}^t \rangle + \sum_{j \in \mathcal{N}(i)} \frac{\rho_i}{2} \|x_{i,j} - z_j^t\|^2.$$

However, this subproblem is hard, especially when f_i is nonconvex. To handle nonconvex objectives, we adopt an alternative solution [78, 111] to this subproblem

through the following first-order approximation of f_i at $\mathbf{z}^t$:

$$\begin{aligned}\mathbf{x}_i^{t+1} &\approx \arg\min_{\mathbf{x}_i} f_i(\mathbf{z}^t) + \langle \nabla f_i(\mathbf{z}^t), \mathbf{x}_i - \mathbf{z}^t \rangle \\ &\quad + \sum_{j \in \mathcal{N}(i)} \left(\langle y_{i,j}^t, x_{i,j} - z_j^t \rangle + \frac{\rho_i}{2} \|x_{i,j} - z_j^t\|^2 \right) \\ &= \mathbf{z}^t - \frac{\nabla f_i(\mathbf{z}^t) + \mathbf{y}_i^t}{\rho_i}, \end{aligned} \tag{6.6}$$

where (6.6) can be readily obtained by setting the partial derivative w.r.t. $\mathbf{x}_i$ to zero.

The above full-vector update on $\mathbf{x}_i$ is equivalent to the following block-wise updates on each block $x_{i,j}$ by worker i:

$$x_{i,j}^{t+1} = z_j^t - \frac{\nabla_j f_i(\mathbf{z}^t) + y_{i,j}^t}{\rho_i}, \tag{6.7}$$

where $\nabla_j f_i(\mathbf{z}^{t+1})$ is the partial derivative of f_i w.r.t. z_j. Furthermore, the dual variable blocks $y_{i,j}$ can also be updated in a block-wise fashion as follows:

$$y_{i,j}^{t+1} = y_{i,j}^t + \rho_i (x_{i,j}^{t+1} - z_j^t). \tag{6.8}$$

Note that in fact, each f_i only depends on a part of $\mathbf{z}^t$ and thus each worker i only needs to pull the relevant blocks z_j for $j \in \mathcal{N}(i)$. Again, we put the full vector $\mathbf{z}$ in $f_i(\cdot)$ just to simplify notation.

On the server side, server j will update z_j based on the newly updated $x_{i,j}^{t+1}$, $y_{i,j}^{t+1}$ received from all workers i such that $i \in \mathcal{N}(j)$. Again, the $\mathbf{z}$ update in the basic synchronous distributed ADMM can be rewritten into the following block-wise format (with a regularization term introduced):

$$\begin{aligned} z_j^{t+1} &= \arg\min_{z_j \in \mathcal{X}_j} h_j(z_j) + \frac{\gamma}{2} \|z_j - z_j^t\|^2 \\ &\quad + \sum_{i \in \mathcal{N}(j)} \left(\langle y_{i,j}^{t+1}, x_{i,j}^{t+1} - z_j \rangle + \frac{\rho_i}{2} \|x_{i,j}^{t+1} - z_j\|^2 \right) \\ &= \mathbf{prox}_{\mu_j h_j} \left(\frac{\gamma z_j^t + \sum_{i \in \mathcal{N}(j)} w_{i,j}^t}{\gamma + \sum_{i \in \mathcal{N}(j)} \rho_i} \right), \end{aligned} \tag{6.9}$$

where $w_{i,j}^{t+1}$ is defined as

$$w_{i,j}^{t+1} := \rho_i x_{i,j}^{t+1} + y_{i,j}^{t+1}. \tag{6.10}$$

Furthermore, the regularization term $\frac{\gamma}{2} \|z_j - z_j^t\|^2$ is introduced to stabilize the results, which will be helpful in the asynchronous case.

When updating z_j, the constant μ of the proximal operator is given by:

$$\mu_j := \frac{1}{\sum_{i\in\mathcal{N}(j)}\rho_i}.$$

Now it is clear that it is sufficient for worker i to send $w_{i,j}^t$ to server j in iteration t.

6.3.2 Block-wise Asynchronous ADMM

We now take one step further to present a *block-wise asynchronous distributed* ADMM algorithm. In the asynchronous algorithm, each worker i will use a local iteration t to keep track of how many times $\mathbf{x}_i$ has been updated, although different workers may be in different iterations, due to random delays in computation and communication.

Let us first focus on a particular worker i at its local iteration t. Due to potential communication delay and reading/writing conflict, we do not expect worker i to download $\mathbf{z}^t$, since this requires synchronization among all workers. Instead, worker i can directly download all blocks from servers without waiting for others or data consistency checking. Therefore, block z_j downloaded from server j may have a delay and we denote $\tilde{z}_j^t$ as the *latest copy* of z_j^t on server j. After that, worker i will perform the following updates on the latest received $\tilde{z}_j^t$:

$$x_{i,j}^{t+1} = \tilde{z}_j^t - \frac{\nabla_j f_i(\tilde{\mathbf{z}}^t) + y_{i,j}^t}{\rho_i}, \tag{6.11}$$

$$y_{i,j}^{t+1} = y_{i,j}^t + \rho_i(x_{i,j}^{t+1} - \tilde{z}_j^t), \tag{6.12}$$

$$w_{i,j}^{t+1} = \rho x_{i,j}^{t+1} + y_{i,j}^{t+1}. \tag{6.13}$$

At the end of local updating, worker i will send $w_{i,j}^{t+1}$ to server j as a updating request, and leave it done by server j. Now we turn to focus on how the server side process workers' updating requests. We use $\tilde{w}_{i',j}^t$ to denote a server's copy of $w_{i',j}^t$ for all $i' \in \mathcal{N}(j)$. When a worker, say worker i, finishes its updating, server j will receive $w_{i,j}^{t+1}$ from worker i, and it will perform the following updates:

$$\tilde{w}_{i,j}^{t+1} = \begin{cases} w_{i,j}^{t+1}, & \text{if received } w_{i,j} \text{ from worker } i, \\ \tilde{w}_{i,j}^t, & \text{otherwise.} \end{cases} \tag{6.14}$$

$$\tilde{z}_j^{t+1} = \mathbf{prox}_{\mu_j h_j}\left(\frac{\gamma\tilde{z}_j^t + \sum_{i\in\mathcal{N}(j)}\tilde{w}_{i,j}^{t+1}}{\gamma + \sum_{i\in\mathcal{N}(j)}\rho_i}\right), \tag{6.15}$$

where the regularization coefficient $\gamma > 0$ helps to stabilize convergence in the asynchronous execution with random delays.

Algorithm 6 AsyBADMM: Block-wise Asynchronous ADMM

Each **worker** i asynchronously performs:

pull $\mathbf{z}^0$ to initialize $\mathbf{x}^0 = \mathbf{z}^0$
initialize $\mathbf{y}^0$ as the zero vector.
for $t = 0$ **to** $T - 1$ **do**
    select an index $j_t \in \mathcal{N}(i)$.
    compute gradient $\nabla_{j_t} f_i(\tilde{\mathbf{z}}^t)$.
    update x_{i,j_t}^{t+1} and y_{i,j_t}^{t+1} by (6.11) and (6.12).
    push w_{i,j_t}^{t+1} as defined by (6.10) to server j_t.
    pull the current models $\tilde{\mathbf{z}}^{t+1}$ from servers.
end for

Each **server** j asynchronously performs:

initialize $\tilde{z}_j^0$ and $\tilde{w}_{i,j}^0$ for all $i \in \mathcal{N}(j)$.
initialize $t = 0$.
while not converge **do**
    for $i = 1$ **to** N **do**
        if worker i finishes its local iteration **then**
            receive $w_{i,j}^t$ from it.
            update server's copy $\tilde{w}_{i,j}^{t+1} \leftarrow w_{i,j}^t$.
        else
            $\tilde{w}_{i,j}^{t+1} \leftarrow \tilde{w}_{i,j}^t$.
        end if
    end for
    update z_j^{t+1} by (6.15)
    $t \leftarrow t + 1$.
end while

Algorithm 6 describes the entire block-wise asynchronous distributed ADMM. Note that in Algorithm 6, the block j is randomly and independently selected from $\mathcal{N}(i)$ according to a uniform distribution, which is common in practice. Due to the page limit, we only consider the cyclic coordinate selection scheme, and we refer readers to other options including random selection rule, Gauss-Seidel and Gauss-Southwell block selection in the literature, *e.g.*, [103]. In addition, the iteration in Algorithm 6 is counted on the worker side for implementation purpose. However, workers may have different local iterations and this makes analysis difficult. In the next section, we will formally define iterations for convergence analysis and provide our theoretical convergence guarantees. Finally, updating z_j is quite flexible in Algorithm 6, once server j collects a number of $w_{i,j}^t$'s from workers. This means, server j can make one update for multiple received updates.

We put a few remarks on implementation issues at the end of this section to characterize key features of our proposed block-wise asynchronous algorithm, which differs from full-vector updates in the literature [78]. *Firstly*, model parameters are stored in blocks, so different workers can update different blocks asynchronously in parallel, which takes advantage of the popular Parameter Server architecture. *Secondly*, workers can pull $\mathbf{z}$ while others are updating some blocks, enhancing concurrency. *Thirdly*, in our implementation, workers will compute both gradients and local variables. In contrast, in the full-vector ADMM [78], workers are only responsible for computing gradients, therefore all previously computed and transmitted $\tilde{w}_{i,j}$ must be cached on servers with non-negligible memory overhead.

6.4 Convergence Analysis

In this section, we provide convergence analysis for our proposed algorithm. To facilitate the analysis of Algorithm 6, we rewrite it in an equivalent global view (from the server's perspective), as described in Algorithm 7. In this algorithm, only the global counter t is used to keep track of how many times the model $\mathbf{z}$ has been updated. Note that such iteration counter is for analysis purpose only and is *not* required by workers to calculate their own variables $x_{i,j}$ and $y_{i,j}$. Also, we assume that worker i will select a server $j \in \mathcal{N}(i)$ in a cyclic round.

Algorithm 7 AsyBADMM (analyzed algorithm)

initialize $\mathbf{z}^0$, split it into M blocks and distribute to servers.
initialize $\tilde{w}_{i,j}^0 = z_j^0$ for all $(i,j) \in \mathcal{E}$.
for $t = 0$ **to** $T-1$ **do**
  for $j = 1$ **to** M **in parallel do**
    pick a set $\mathcal{C}_j^t \in \mathcal{N}(j)$.
    for $i \in \mathcal{N}(j)$ **do**
      if $i \in \mathcal{C}_j^t$ **then**
        pull the current models $\tilde{\mathbf{z}}^t$ from servers.
        update $x_{i,j}^{t+1}$ and $y_{i,j}^{t+1}$ according to (6.11) and (6.12), respectively.
        calculate $w_{i,j}^{t+1}$ according to (6.13).
        $\tilde{w}_{i,j}^{t+1} \leftarrow w_{i,j}^{t+1}$.
      else
        $\tilde{w}_{i,j}^{t+1} \leftarrow \tilde{w}_{i,j}^{t}$.
      end if
    end for
    update z_j^{t+1} according to (6.15).
  end for
end for

6.4.1 Assumptions and Metrics

Our convergence analysis is based on the following assumptions, which is similar to the assumptions made in previous work on the analysis of asynchronous ADMM, *e.g.*, [78].

Assumption 6.1 (Block Lipschitz Continuity)**.** *For all $(i,j) \in \mathcal{E}$, there exists a positive constant $L_{i,j} > 0$ such that*

$$\|\nabla_j f_i(\mathbf{x}) - \nabla_j f_i(\mathbf{z})\| \leq L_{i,j}\|x_j - z_j\|, \forall \mathbf{x}, \mathbf{z} \in \mathbb{R}^d.$$

Assumption 6.2 (Bounded from Below)**.** *Each function $f_i(\mathbf{x})$is bounded below, i.e., there exists a finite number $\underline{f} > -\infty$ where $\underline{f}$ denotes the optimal objective value of problem* (6.5)*.*

Assumption 6.3 (Bounded Delay)**.** *The total delay of each link $(i,j) \in \mathcal{E}$ is bounded with the constant of $T_{i,j}$ for each pair of worker i and server j. Formally, there is an integer $0 \leq \tau \leq T$, such that $\tilde{z}_j^t = z_j^{t-\tau}$ for all $t > 0$. This should also hold for $\tilde{w}_{i,j}$.*

Since the objective is nonconvex and nonsmooth, AsyBADMM may converge to a critical point. To indicate convergence to stationary points of a nonconvex

but smooth objective function, a commonly used metric is the squared norm of gradients, *e.g.*, [24, 75]. To handle the nonsmoothness of $h(\cdot)$, we consider another metric that studied in [111]:

$$P(\mathbf{X}^t, \mathbf{Y}^t, \mathbf{z}^t) := \|\mathbf{z}^t - \hat{\mathbf{z}}^t\|^2 + \sum_{(i,j)\in\mathcal{E}} \|\nabla_{x_{i,j}} L(\mathbf{X}^t, \mathbf{Y}^t, \mathbf{z}^t)\|^2 + \sum_{(i,j)\in\mathcal{E}} \|x_{i,j} - z_j\|^2, \tag{6.16}$$

where $\hat{\mathbf{z}}^t = (\hat{z}_1^t, \ldots, \hat{z}_M^t)$ is defined as

$$\hat{z}_j^t := \mathbf{prox}_h(z_j^t - \nabla_{z_j}(L(\mathbf{X}^t, \mathbf{Y}^t, \mathbf{z}^t) - h(\mathbf{z}^t))). \tag{6.17}$$

The first term in $P(\mathbf{X}^t, \mathbf{Y}^t, \mathbf{z}^t)$ is the distance between $\mathbf{z}^t$ and its gradient projection $\hat{\mathbf{z}}^t$. This is in the same spirit of gradient mapping $\mathcal{P}(\mathbf{z}^t)$ defined in (1.4), Chapter 1. The second term shows the norm of partial gradient w.r.t. $x_{i,j}$'s, and the last term measures how the consensus constraints $x_{i,j} = z_j$ are satisfied. It is clear that when $P(\mathbf{X}^t, \mathbf{Y}^t, \mathbf{z}^t) \to 0$, the corresponding sequence converges to a critical point. Therefore, we will focus on ϵ-accuracy in our convergence analysis by looking at how many iterations needed to achieve $P(\mathbf{X}^t, \mathbf{Y}^t, \mathbf{z}^t) \leq \epsilon$.

6.4.2 Main Result

The following Theorem 6.4.1 indicates that Algorithm 6 converges to a critical point under suitable choices of hyper-parameters.

Theorem 6.4.1. *Suppose that Assumptions 6.1-6.3 hold. Moreover, for all i and j, the penalty parameter ρ_i and γ are chosen to be sufficiently large such that:*

$$\infty > L(\mathbf{X}^0, \mathbf{Y}^0, \mathbf{z}^0) - \underline{f} \geq 0 \tag{6.18}$$

$$\alpha_{i,j} := (\gamma + \rho_i) - \left(\frac{7L_{i,j}}{2\rho_i^2} + \frac{1}{\rho_i} + \frac{1}{2}\right) L_{i,j}^2 (T+1)^2 - \left(\frac{(2+\rho_i)T^2}{2}\right) > 0, \tag{6.19}$$

$$\beta_{i,j} := \frac{\rho_i}{4} - 3L_{i,j} > 0. \tag{6.20}$$

Then the following is true for Algorithm 6:

1. *Algorithm 6 converges in the following sense:*

$$\lim_{t\to\infty} \|z_j^{t+1} - z_j^t\| = 0, \quad \forall j = 1, \ldots, M, \tag{6.21a}$$

$$\lim_{t\to\infty} \|x_{i,j}^{t+1} - x_{i,j}^t\| = 0, \quad \forall (i,j) \in \mathcal{E}, \tag{6.21b}$$

$$\lim_{t\to\infty} \|y_{i,j}^{t+1} - y_{i,j}^t\| = 0, \quad \forall (i,j) \in \mathcal{E}. \tag{6.21c}$$

2. *For each worker i and server j, denote the limit points of $\{x_{i,j}^t\}, \{y_{i,j}^t\}$, and $\{z_j^t\}$ by $x_{i,j}^*, y_{i,j}^*$ and z_j^*, respectively. Then these limit points satisfy the following conditions:*

$$\nabla_j f_i(\mathbf{x}_i^*) + y_{i,j}^* = 0, \quad \forall (i,j) \in \mathcal{E}, \tag{6.22a}$$

$$\sum_{j \in \mathcal{N}(i)} y_{i,j}^* \in \partial h_j(z_j^*), \quad \forall j = 1, \ldots, M, \tag{6.22b}$$

$$x_{i,j}^* = z_j^* \in \mathcal{X}_j, \quad \forall (i,j) \in \mathcal{E}. \tag{6.22c}$$

When sets $\mathcal{X}_j$ are compact, the sequence of iterates generated by Algorithm 6 converges to stationary points.

3. *For some $\epsilon > 0$, let $\mathcal{T}(\epsilon)$ denote the iteration that achieves the following:*

$$\mathcal{T}(\epsilon) = \min\{t | P(\mathbf{X}^t, \mathbf{Y}^t, \mathbf{z}^t) \leq \epsilon, t \geq 0\}.$$

Then there exists some constant $C > 0$ such that

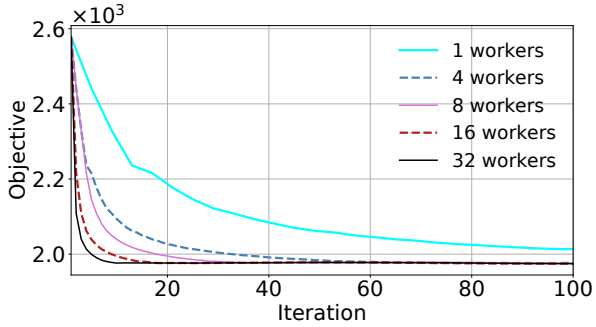
$$\mathcal{T}(\epsilon) \leq \frac{C(L(\mathbf{X}^0, \mathbf{Y}^0, \mathbf{z}^0) - \underline{f})}{\epsilon}, \tag{6.23}$$

where $\underline{f}$ is defined in Assumption 6.2.

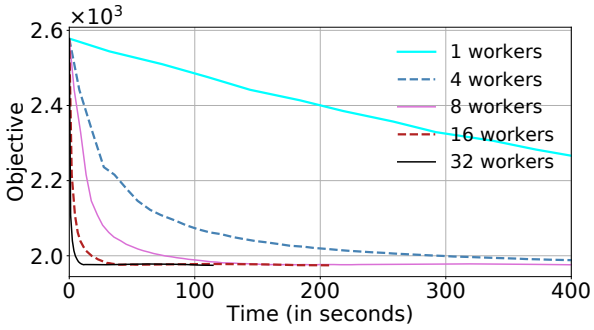
Due to the nonconvex objective function $f_i(\mathbf{x}_i)$, no guarantee of global optimality is possible in general. The parameter ρ_i acts like the learning rate hyper-parameter in gradient descent: a large ρ_i slows down the convergence and a smaller one can speed it up. The term γ is associated with the delay bound T. In the synchronous case, we can set $\gamma = 0$; otherwise, to guarantee convergence, γ should be increased as the maximum allowable delay T increases.

6.5 Experiments

We now show how our algorithm can be used to solve the challenging nonconvex nonsmooth problems in machine learning. We will show how AsyBADMM exhibits a near-linear speedup as the number of workers increases. We use a cluster of 18 instances of type `c4.large` on Amazon EC2. This type of instances has 2 CPU cores and at least 3.75 GB RAM, running 64-bit Ubuntu 16.04 LTS (HVM). Each server and worker process uses up to 2 cores. In total, our deployment uses 36 CPU cores and 67.5 GB RAM. Two machines serve as server nodes, while the other 16



(a) iteration vs. objective



(b) time vs. objective

Figure 6.1: Convergence of AsyBADMM on the sparse logistic regression problem.

machines serve as worker nodes. Note that we treat one core as a computational node (either a worker or server node).

Setup: In this experiment, we consider the sparse logistic regression problem:

$$\begin{aligned} \min_{\mathbf{x}} \quad & \sum_{i=1}^{N} f_i(\mathbf{x}) + \lambda \|\mathbf{x}\|_1 \\ \text{s.t.} \quad & f_i(\mathbf{x}) = \sum_{j \in \mathcal{S}_i} \log(1 + \exp(-b_j \langle \mathbf{a}_j, \mathbf{x} \rangle)) \\ & \|\mathbf{x}\|_\infty \leq C. \end{aligned} \tag{6.24}$$

where the constant C is used to clip out some extremely large values for robustness. The ℓ_1-regularized logistic regression is one of the most popular algorithms used for large scale risk minimization. We consider a public sparse text dataset `KDDa` [1]. This dataset has more than 8 million samples, 20 million features, and 305 million nonzero entries. To show the advantage of parallelism, we set up five experiments with 1, 4, 8, 16 and 32 nodes, respectively. In each experiment, the whole dataset will be evenly split into several smaller parts, and each node only has access to its local dataset $\mathcal{S}_i$.

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

Table 6.1: Running time (in seconds) for iterations k and worker count.

Workers p	$k = 20$	$k = 50$	$k = 100$	Speedup
1	1404	3688	6802	1.0
4	363	952	1758	3.87
8	177	466	859	7.92
16	86	226	417	16.31
32	47	124	228	29.83

We implement our algorithm on the `ps-lite` framework [16], which is a lightweight implementation of Parameter Server architecture. It supports Parameter Server for multiple devices in a single machine, and multiple machines in a cluster. This is the back end of `kvstore` API of the deep learning framework MXNet [9]. We follow the rule of cycling coordinate selection through the coordinates of $\mathbf{x}$ and updating each in turns, restarting at a random coordinate after each cycle.

Results: We set the hyper-parameter $\gamma = 0.01$ and the clip threshold constant as $C = 10^4$, and the penalty parameter $\rho_i = 100$ for all i. Fig. 6.1(a) and Fig. 6.1(b) show the convergence behavior of our proposed algorithm in terms of objective function values, which are normalized by the number of total samples in the dataset. From the figures, we can clearly observe the convergence of our proposed algorithm. This observation confirms that asynchrony with tolerable delay can still lead to convergence.

To further analyze the parallelism in AsyBADMM, we measure the speedup by the relative time for p workers to perform k iterations, i.e., Speedup of p workers $= \frac{T_k(1)}{T_k(p)}$, where $T_k(p)$ is the time it takes for p workers to perform k iterations of optimization. Fig. 6.1(b) illustrates the running time comparison and Table 6.1 shows that AsyBADMM actually achieves near-linear speedup.

6.6 Preliminaries

We denote $j_t(i)$ as the block index selected by worker i at iteration t, and $j_t(i) = \emptyset$ when worker i is not performed. We put all such $j_t(i)$ to form a vector $J_t := (j_t(1), \ldots, j_t(M))$. Since selecting a block index is done by randomly selection rule, J_t here is a random variable.

In the following of this chapter, we use the following additional notations used

to bound $x_{i,j}$, $y_{i,j}$ and z_j for a particular worker i and a server j s.t. $(i,j) \in \mathcal{E}$:

$$l_i(\mathbf{x}_i, \mathbf{z}, \mathbf{y}_i) := f_i(\mathbf{x}_i) + \sum_{j \in \mathcal{N}(i)} \langle y_{i,j}, x_{i,j} - z_j \rangle + \sum_{j \in \mathcal{N}(i)} \frac{\rho_i}{2} \|x_{i,j} - z_j\|^2 \quad (6.25)$$

$$u_i(\mathbf{x}_i, \tilde{\mathbf{z}}; \mathbf{z}, \mathbf{y}_i) := f_i(\tilde{\mathbf{z}}) + \sum_{j \in \mathcal{N}(i)} \langle \nabla_j f_i(\tilde{\mathbf{z}}), x_{i,j} - \tilde{z}_j \rangle + \sum_{j \in \mathcal{N}(i)} \langle y_{i,j}, x_{i,j} - z_j \rangle + \sum_{j \in \mathcal{N}(i)} \frac{\rho_i}{2} \|x_{i,j} - z_j\|^2 \quad (6.26)$$

$$\tilde{u}_i(\mathbf{x}_i, \tilde{\mathbf{z}}, \mathbf{y}_i) := f_i(\tilde{\mathbf{z}}) + \sum_{j \in \mathcal{N}(i)} \langle \nabla_j f_i(\tilde{\mathbf{z}}), x_{i,j} - \tilde{z}_j \rangle + \sum_{j \in \mathcal{N}(i)} \langle y_{i,j}, x_{i,j} - \tilde{z}_j \rangle + \sum_{j \in \mathcal{N}(i)} \frac{\rho_i}{2} \|x_{i,j} - \tilde{z}_j\|^2 \quad (6.27)$$

6.6.1 Auxiliary Lemmas

Lemma 6.6.1. *Suppose Assumption 6.1-6.3 are satisfied. Then we have*

$$\|y_{i,j}^{t+1} - y_{i,j}^t\|^2 \leq L_{i,j}^2 (T+1) \sum_{t'=0}^{T} \|z_j^{t+1-t'} - z_j^{t-t'}\|^2. \quad (6.28)$$

Proof. For simplicity, we say (i,j) is performed at iteration t when worker i is updating block j at iteration t. If the updating (i,j) is not performed at iteration t, this inequality holds in trivial, as $y_{i,j}^{t+1} = y_{i,j}^t$. So we only consider the case that (i,j) is performed at iteration t. Note that in this case, we have

$$\nabla_j f_i(\tilde{\mathbf{z}}^{t+1}) + y_{i,j}^t + \rho_i (x_{i,j}^{t+1} - \tilde{z}_j^{t+1}) = 0. \quad (6.29)$$

Since $y_{i,j}^{t+1} = y_{i,j}^t + \rho_i (x_{i,j}^{t+1} - \tilde{z}_j^{t+1})$, we have

$$\nabla_j f_i(\tilde{\mathbf{z}}^{t+1}) + y_{i,j}^{t+1} = 0. \quad (6.30)$$

Therefore, we have

$$\begin{aligned} \|y_{i,j}^{t+1} - y_{i,j}^t\| &= \|\nabla_j f_i(\tilde{\mathbf{z}}^{t+1}) - \nabla_j f_i(\tilde{\mathbf{z}}^t)\| \\ &\leq L_{i,j} \|\tilde{z}_j^{t+1} - \tilde{z}_j^t\|. \end{aligned}$$

Since the actual updating time for $\tilde{z}_j^{t+1}$ should be in $\{t+1, t, \ldots, t+1-T\}$, and for $\tilde{z}_j^t$ in $\{t, \ldots, t-T\}$, then we have

$$\|y_{i,j}^{t+1} - y_{i,j}^t\|^2 \leq L_{i,j}^2 (T+1) \sum_{t'=0}^{T} \|z_j^{t+1-t'} - z_j^{t-t'}\|^2, \quad (6.31)$$

which proves the lemma. □

Lemma 6.6.2. *At iteration t, we have*

$$\nabla_j f_i(\tilde{\mathbf{z}}^{t+1}) + y_{i,j}^t + \rho_i(x_{i,j}^{t+1} - z_j^{t+1}) = \rho_i(\tilde{z}_j^{t+1} - z_j^{t+1}). \tag{6.32}$$

Proof. Updating $x_{i,j}^{t+1}$ is performed as follows

$$x_{i,j}^{t+1} = \arg\min_{x_{i,j}} f_i(\tilde{\mathbf{z}}^{t+1}) + \langle \nabla_j f_i(\tilde{\mathbf{z}}^{t+1}), x_{i,j} - \tilde{z}_j^{t+1}\rangle + \langle y_{i,j}^t, x_{i,j} - z_j^{t+1}\rangle + \frac{\rho_i}{2}\|x_{i,j} - z_j^{t+1}\|^2.$$

Thus, we have

$$\nabla_j f_i(\tilde{\mathbf{z}}^{t+1}) + y_{i,j}^t + \rho_i(x_{i,j}^{t+1} - \tilde{z}_j^{t+1}) = 0,$$

And therefore

$$\nabla_j f_i(\tilde{\mathbf{z}}^{t+1}) + y_{i,j}^t + \rho_i(x_{i,j}^{t+1} - z_j^{t+1}) = \rho_i(\tilde{z}_j^{t+1} - z_j^{t+1}).$$

□

6.7 Proof of Theorem 6.4.1

When trying to bound the gap between two consecutive Augmented Lagrangian values, we decompose it into three smaller steps, namely, updating $\mathbf{X}$, $\mathbf{Y}$ and $\mathbf{z}$:

$$\begin{aligned} & L(\mathbf{X}^{t+1}, \mathbf{Y}^{t+1}, \mathbf{z}^{t+1}) - L(\mathbf{X}^t, \mathbf{Y}^t, \mathbf{z}^t) \\ &= \underbrace{L(\mathbf{X}^{t+1}, \mathbf{Y}^t, \mathbf{z}^t) - L(\mathbf{X}^t, \mathbf{Y}^t, \mathbf{z}^t)}_{\text{updating } \mathbf{X}^t} \\ &\quad + \underbrace{L(\mathbf{X}^{t+1}, \mathbf{Y}^{t+1}, \mathbf{z}^t) - L(\mathbf{X}^{t+1}, \mathbf{Y}^t, \mathbf{z}^t)}_{\text{updating } \mathbf{Y}^t} \\ &\quad + \underbrace{L(\mathbf{X}^{t+1}, \mathbf{Y}^{t+1}, \mathbf{z}^{t+1}) - L(\mathbf{X}^{t+1}, \mathbf{Y}^{t+1}, \mathbf{z}^t)}_{\text{updating } \mathbf{z}^t}. \end{aligned} \tag{6.33}$$

The proof of Theorem 6.4.1 relies on the following two key lemmas:

Lemma 6.7.1. *Suppose Assumption 6.1-6.3 are satisfied. Then we have*

$$\begin{aligned} & L(\mathbf{X}^{t+1}, \mathbf{Y}^{t+1}, \mathbf{z}^{t+1}) - L(\mathbf{X}^1, \mathbf{Y}^1, \mathbf{z}^1) \\ \leq & -\sum_{d=1}^{t} \sum_{(i,j)\in\mathcal{E}} \beta_{i,j}\|x_{i,j}^{t+1} - x_{i,j}^t\|^2 - \sum_{d=1}^{t} \sum_{(i,j)\in\mathcal{E}} \alpha_{i,j}\|z_j^{t+1} - z_j^t\|^2, \end{aligned} \tag{6.34}$$

where

$$\alpha_{i,j} := (\gamma + \rho_i) - \left(\frac{7L_{i,j}}{2\rho_i^2} + \frac{1}{\rho_i} + \frac{1}{2}\right) L_{i,j}^2 (T+1)^2 - \left(\frac{(2+\rho_i)T^2}{2}\right), \tag{6.35}$$

$$\beta_{i,j} := \frac{\rho_i}{4} - 3L_{i,j}. \tag{6.36}$$

Lemma 6.7.2. *Suppose that Assumption 6.1-6.3 hold. Then the sequence of solutions* $\{\mathbf{X}^t, \mathbf{Y}^t, \mathbf{z}^t\}$ *satisfies*

$$\lim_{t\to\infty} L(\mathbf{X}^t, \mathbf{Y}^t, \mathbf{z}^t) \geq \underline{f} - \operatorname{diam}^2(\mathcal{X}) \sum_{(i,j)\in\mathcal{E}} \frac{L_{i,j}}{2} > -\infty. \tag{6.37}$$

Given these two lemmas, we are ready to prove Theorem 6.4.1.

Proof of Theorem 6.4.1. From Lemma 6.7.1, we must have, as $t \to \infty$,

$$\|z_j^{t+1} - z_j^t\| \to 0, \|x_{i,j}^{t+1} - x_{i,j}^t\| \to 0. \tag{6.38}$$

Given Lemma 6.6.1, we have $y_{i,j}^{t+1} - y_{i,j}^t \to 0$, which proves (6.22c), the first part.

For the second part, we have the following inequality from the optimality condition of (6.15):

$$0 \in \partial h_j(z_j^{t+1}) - \sum_{i\in\mathcal{N}(j)} \left(y_{i,j}^{t+1} + \rho_i(x_{i,j}^{t+1} - z_j^{t+1}) + \gamma(z_j^t - z_j^{t+1}) \right). \tag{6.39}$$

From (6.21a) and (6.22c), we have

$$0 \in \partial h_j(z_j^*) - \sum_{i\in\mathcal{N}(j)} y_{i,j}^*, \tag{6.40}$$

which proves (6.22b). Finally, from the optimality condition in (6.15), we have (6.11) which implies (6.22a), the second part of the theorem.

We now turn to prove the last part. Let $L'(\mathbf{X}, \mathbf{Y}, \mathbf{z}) := L(\mathbf{X}, \mathbf{Y}, \mathbf{z}) - h(\mathbf{z})$, which excludes $h(\mathbf{z})$ from $L(\mathbf{X}, \mathbf{Y}, \mathbf{z})$. Denote its j-th component as

$$l'(\mathbf{X}, \mathbf{Y}, z_j) := \sum_{i\in\mathcal{N}(j)} \left(f_i(\mathbf{x}_i) + \langle y_{i,j}, x_{i,j} - z_j \rangle + \frac{\rho_i}{2}\|x_{i,j} - z_j\|^2 \right).$$

Then, we have

$$\begin{aligned} z_j - \nabla_{z_j} l'(\mathbf{X}, \mathbf{Y}, z_j) &= z_j - \sum_{i\in\mathcal{N}(j)} y_{i,j} - \sum_{i\in\mathcal{N}(j)} \rho_i(x_{i,j} - z_j) \\ &= z_j - \sum_{i\in\mathcal{N}(j)} \rho_i(z_j - x_{i,j} - \frac{y_{i,j}}{\rho_i}). \end{aligned}$$

Therefore, we have

$$
\begin{aligned}
&\|z_j^t - \mathbf{prox}_h(z_j^t - \nabla_{z_j} l'(\mathbf{X}^t, \mathbf{Y}^t, z_j^t))\| && (6.41)\\
&\le \|z_j^t - z_j^{t+1} + z_j^{t+1} - \mathbf{prox}_h(z_j^t - \nabla_{z_j} l'(\mathbf{X}^t, \mathbf{Y}^t, z_j^t))\| \\
&\le \|z_j^t - z_j^{t+1}\| + \left\| z_j^{t+1} - \mathbf{prox}_h\left(z_j - \sum_{i\in\mathcal{N}(j)} \rho_i (z_j^t - x_{i,j}^t - \frac{y_{i,j}^t}{\rho_i}) \right) \right\| \\
&\le \|z_j^t - z_j^{t+1}\| + \left\| \mathbf{prox}_h\left(z_j^{t+1} - \sum_{i\in\mathcal{N}(j)} \rho_i (z_j^{t+1} - x_{i,j}^{t+1} - \frac{y_{i,j}^{t+1}}{\rho_i}) + \gamma(z_j^{t+1} - z_j^t) \right) \right. \\
&\qquad\qquad \left. - \mathbf{prox}_h\left(z_j^t - \sum_{i\in\mathcal{N}(j)} \rho_i (z_j^t - x_{i,j}^t - \frac{y_{i,j}^t}{\rho_i}) \right) \right\| && (6.42)\\
&\le \left(2 + \gamma + \sum_{i\in\mathcal{N}(j)} \rho_i \right) \|z_j^t - z_j^{t+1}\|, && (6.43)
\end{aligned}
$$

where (6.42) is from the optimality in (6.15) as

$$
z_j^{t+1} = \mathbf{prox}_h(z_j^{t+1} - \sum_{i\in\mathcal{N}(j)} \rho_i (z_j^{t+1} - x_{i,j}^{t+1} - \frac{y_{i,j}^{t+1}}{\rho_i}) + \gamma(z_j^{t+1} - z_j^t)),
$$

and (6.43) is from the firm nonexpansiveness property of proximal operator. Then, by the update rule of $x_{i,j}^{t+1}$, if $x_{i,j}$ is selected to update at iteration t, we have

$$
\begin{aligned}
\|\nabla_{x_{i,j}} L(\mathbf{X}^t, \mathbf{Y}^t, \mathbf{z}^t)\|^2 &= \|\nabla_j f_i(\mathbf{x}_i^t) + \rho_i(x_{i,j}^t - z_j^t + \frac{y_{i,j}^t}{\rho_i})\|^2 \\
&= \|\nabla_j f_i(\mathbf{x}_i^t) - \nabla_j f_i(\tilde{\mathbf{z}}^t) + (y_{i,j}^t - y_{i,j}^{t-1}) + \rho_i(\tilde{z}_j^t - z_j^t)\|^2 \\
&\le 3\|\nabla_j f_i(\mathbf{x}_i^t) - \nabla_j f_i(\tilde{\mathbf{z}}^t)\|^2 + 3\|y_{i,j}^t - y_{i,j}^{t-1}\|^2 + 3\|\rho_i(\tilde{z}_j^t - z_j^t)\|^2 \\
&\le 3L_{i,j}^2 \|x_{i,j}^t - \tilde{z}_j^t\|^2 + 3\|y_{i,j}^t - y_{i,j}^{t-1}\|^2 + 3\rho_i^2 \|\tilde{z}_j^t - z_j^t\|^2 \\
&\le 3(L_{i,j}^2 + \rho_i^2)\|x_{i,j}^t - \tilde{z}_j^t\|^2 + 3\rho_i^2 \|\tilde{z}_j^t - z_j^t\|^2,
\end{aligned}
$$

which implies that there must exist two positive constant $\sigma_1 > 0$ and $\sigma_2 > 0$ such that

$$
\sum_{(i,j)\in\mathcal{E}} \|\nabla_{x_{i,j}} L(\mathbf{X}^t, \mathbf{Y}^t, \mathbf{z}^t)\|^2 \le \sum_{(i,j)\in\mathcal{E}} \sigma_1 \|x_{i,j}^t - \tilde{z}_j^t\|^2 + \sum_{(i,j)\in\mathcal{E}} \sigma_2 \sum_{t'=0}^{T-1} \|z_j^{t-t'} - z_j^{t-t'-1}\|. \tag{6.44}
$$

The last step is to estimate $\|x_{i,j}^t - z_j^t\|$, which can be done as follows:

$$
\|x_{i,j}^t - z_j^t\|^2 \le \|x_{i,j}^t - \tilde{z}_j^t\|^2 + \|\tilde{z}_j^t - z_j^t\|^2 \tag{6.45}
$$

$$
\sum_{(i,j)\in\mathcal{E}} \|x_{i,j}^t - z_j^t\|^2 \le \sum_{(i,j)\in\mathcal{E}} (\|x_{i,j}^t - \tilde{z}_j^t\|^2 + \|\tilde{z}_j^t - z_j^t\|^2) \tag{6.46}
$$

Combining (6.43), (6.44) and (6.46), and summing up $t = 0, \ldots, T-1$, we have

$$\sum_{t=0}^{T-1} P(\mathbf{X}^t, \mathbf{Y}^t, \mathbf{z}^t) \leq \sum_{t=0}^{T-1} \sum_{(i,j)\in\mathcal{E}} \sigma_3 \|x_{i,j}^t - \tilde{z}_j^t\|^2 + \sigma_4 T \|z_j^{t+1} - z_j^t\|^2. \tag{6.47}$$

From Lemma 6.7.1, we have

$$L(\mathbf{X}^T, \mathbf{Y}^T, \mathbf{z}^T) - L(\mathbf{X}^0, \mathbf{Y}^0, \mathbf{z}^0) \tag{6.48}$$

$$\leq -\sum_{t=0}^{T-1} \sum_{(i,j)\in\mathcal{E}} \beta_i (\|x_{i,j}^{t+1} - \tilde{z}_j^{t+1}\|^2 + \|x_{i,j}^t - z_j^{t+1}\|^2) - \sum_{t=0}^{T-1} \sum_{(i,j)\in\mathcal{E}} \alpha_j \|z_j^{t+1} - z_j^t\|^2 \tag{6.49}$$

$$\leq -\sum_{t=0}^{T-1} \sum_{(i,j)\in\mathcal{E}} \delta_1 \|x_{i,j}^{t+1} - \tilde{z}_j^{t+1}\|^2 + \delta_2 \|z_j^{t+1} - z_j^t\|^2, \tag{6.50}$$

where $\delta_1 := \min_i \beta_i$ and $\delta_2 := \min_j \alpha_j$.Now we can find some $C > 0$, such that the following equation hold:

$$\begin{aligned}\sum_{t=0}^{T-1} P(\mathbf{X}^t, \mathbf{Y}^t, \mathbf{z}^t) &\leq C(L(\mathbf{X}^0, \mathbf{Y}^0, \mathbf{z}^0) - L(\mathbf{X}^T, \mathbf{Y}^T, \mathbf{z}^T)) \\ &\leq C(L(\mathbf{X}^0, \mathbf{Y}^0, \mathbf{z}^0) - \underline{f}),\end{aligned}$$

where the last inequality we have used the fact that $L(\mathbf{X}^t, \mathbf{Y}^t, \mathbf{z}^t)$ is lowered bounded by $\underline{f}$ for all t from Lemma 6.7.2. Let $T = T(\epsilon)$ and we have

$$T(\epsilon) \leq \frac{C(L(\mathbf{X}^0, \mathbf{Y}^0, \mathbf{z}^0) - \underline{f})}{\epsilon}, \tag{6.51}$$

which proves the last part of Theorem 6.4.1. □

6.7.1 Proof of Lemma 6.7.1

We decompose Lemma 6.7.1 into the following three lemmas.

Lemma 6.7.3. *For all worker i, we have*

$$\begin{aligned}&l_i(\mathbf{x}_i^{t+1}, \mathbf{z}^t, \mathbf{y}_i^t) - l_i(\mathbf{x}_i^t, \mathbf{z}^t, \mathbf{y}_i^t) \\ \leq &\sum_{j\in\mathcal{N}(i)} \left[-\left(\frac{\rho_i - 12L_{i,j}}{4}\right) \|x_{i,j}^{t+1} - x_{i,j}^t\|^2 + \rho_i T \sum_{t'=0}^{T-1} \|z_j^{t+1-t'} - z_j^{t-t'}\|^2 \right] \\ &+ \sum_{j\in\mathcal{N}(i)} \frac{7L_{i,j}}{2\rho_i^2} \|y_{i,j}^{t+1} - y_{i,j}^t\|^2\end{aligned} \tag{6.52}$$

Proof. From the block Lipschitz assumption, we have

$$f_i(\mathbf{x}_i^{t+1}) \leq f_i(\tilde{\mathbf{z}}_i^t) + \langle \nabla f_i(\tilde{\mathbf{z}}^t), \mathbf{x}^{t+1} - \tilde{\mathbf{z}}^t \rangle + \sum_{j\in\mathcal{N}(i)} \frac{L_{i,j}}{2} \|x_{i,j}^{t+1} - \tilde{z}_j^t\|^2, \tag{6.53}$$

and thus

$$l_i(\mathbf{x}_i^{t+1}, \mathbf{z}^t, \mathbf{y}_i^t) \leq u_i(\mathbf{x}_i^{t+1}, \tilde{\mathbf{z}}^t; \mathbf{z}^t, \mathbf{y}_i^t) + \sum_{j\in\mathcal{N}(i)} \frac{L_{i,j}}{2}\|x_{i,j}^{t+1} - \tilde{z}_j\|^2. \tag{6.54}$$

On the right hand side of (6.54), we can see that $u_i(\mathbf{x}_i^{t+1}, \tilde{\mathbf{z}}^t; \mathbf{z}^t, \mathbf{y}_i^t)$ is a strongly convex w.r.t. $\mathbf{x}_i^{t+1}$. Then, we have

$$\begin{aligned}
&u_i(\mathbf{x}_i^{t+1}, \tilde{\mathbf{z}}^t; \mathbf{z}^t, \mathbf{y}_i^t) - u_i(\mathbf{x}_i^t, \tilde{\mathbf{z}}^t; \mathbf{z}^t, \mathbf{y}_i^t) \\
\leq&\langle \nabla u_i(\mathbf{x}_i^{t+1}, \tilde{\mathbf{z}}^t; \mathbf{z}^t, \mathbf{y}_i^t), \mathbf{x}_i^{t+1} - \mathbf{x}_i^t\rangle - \sum_{j\in\mathcal{N}(i)} \frac{\rho_i}{2}\|x_{i,j}^{t+1} - x_{i,j}^t\|^2 &(6.55)\\
=&\sum_{j\in\mathcal{N}(i)} \langle \nabla_j f_i(\tilde{\mathbf{z}}^t) + y_{i,j} + \rho_i(\tilde{z}_j^t - z_j^t), x_{i,j}^{t+1} - x_{i,j}^t\rangle - \frac{\rho_i}{2}\|x_{i,j}^{t+1} - x_{i,j}^t\|^2 &(6.56)\\
\overset{(a)}{=}&\sum_{j\in\mathcal{N}(i)} \langle \rho_i(\tilde{z}_j^t - z_j^t), x_{i,j}^{t+1} - x_{i,j}^t\rangle - \frac{\rho_i}{2}\|x_{i,j}^{t+1} - x_{i,j}^t\|^2 &(6.57)\\
\overset{(b)}{\leq}&\sum_{j\in\mathcal{N}(i)} \rho_i\|\tilde{z}_j^t - z_j^t\|^2 - \frac{\rho_i}{4}\|x_{i,j}^{t+1} - x_{i,j}^t\|^2, &(6.58)
\end{aligned}$$

where (a) is from Lemma 6.7.1 and (b) is from the Young's inequality, *i.e.*,

$$\langle a, b\rangle \leq \frac{1}{2\delta}\|a\|^2 + \frac{\delta}{2}\|b\|^2,$$

for all a, b and $\delta > 0$. We will see this inequality in later chapters. Now we turn to estimate the gap between $u_i(\mathbf{x}_i^t, \tilde{\mathbf{z}}^t; \mathbf{z}^t, \mathbf{y}_i^t)$ and $l_i(\mathbf{x}_i^t, \mathbf{z}^t, \mathbf{y}^t)$ as follows:

$$\begin{aligned}
&u_i(\mathbf{x}_i^t, \tilde{\mathbf{z}}^t; \mathbf{z}^t, \mathbf{y}_i^t) - l_i(\mathbf{x}_i^t, \mathbf{z}^t, \mathbf{y}^t) \\
=&f_i(\tilde{\mathbf{z}}^t) + \langle \nabla f_i(\tilde{z}^t), \mathbf{x}_i^t - \tilde{\mathbf{z}}^t\rangle - f_i(\mathbf{x}^t) &(6.59)\\
\leq&\langle \nabla f_i(\tilde{\mathbf{z}^t}) - \nabla f_i(\mathbf{x}^t), \mathbf{x}_i^t - \tilde{\mathbf{z}}^t\rangle + \sum_{j\in\mathcal{N}(i)} \frac{L_{i,j}}{2}\|\tilde{z}_j^t - z_j^t\|^2 &(6.60)\\
\leq&\sum_{j\in\mathcal{N}(i)} \frac{3L_{i,j}}{2}\|\tilde{z}_j^t - x_{i,j}^t\|^2. &(6.61)
\end{aligned}$$

Combining (6.54), (6.58), and (6.61), we have

$$
\begin{aligned}
& l_i(\mathbf{x}_i^{t+1}, \mathbf{z}^t, \mathbf{y}_i^t) - l_i(\mathbf{x}_i^t, \mathbf{z}^t, \mathbf{y}_i^t) \\
\leq & u_i(\mathbf{x}_i^{t+1}, \tilde{\mathbf{z}}^t; \mathbf{z}^t, \mathbf{y}_i^t) + \sum_{j \in \mathcal{N}(i)} \frac{L_{i,j}}{2} \|x_{i,j}^{t+1} - \tilde{z}_j\|^2 && (6.62) \\
\leq & \left(u_i(\mathbf{x}_i^{t+1}, \tilde{\mathbf{z}}^t; \mathbf{z}^t, \mathbf{y}_i^t) - u_i(\mathbf{x}_i^t, \tilde{\mathbf{z}}^t; \mathbf{z}^t, \mathbf{y}_i^t)\right) + \left(u_i(\mathbf{x}_i^t, \tilde{\mathbf{z}}^t; \mathbf{z}^t, \mathbf{y}_i^t) - l_i(\mathbf{x}_i^t, \mathbf{z}^t, \mathbf{y}_i^t)\right) \\
& + \sum_{j \in \mathcal{N}(i)} \frac{L_{i,j}}{2} \|x_{i,j}^{t+1} - \tilde{z}_j^t\|^2 && (6.63) \\
\leq & \sum_{j \in \mathcal{N}(i)} \rho_i \|\tilde{z}_j^t - z_j^t\|^2 - \frac{\rho_i}{4} \|x_{i,j}^{t+1} - x_{i,j}^t\|^2 \\
& + \sum_{j \in \mathcal{N}(i)} \frac{3L_{i,j}}{2} \|\tilde{z}_j^t - x_{i,j}^t\|^2 + \sum_{j \in \mathcal{N}(i)} \frac{L_{i,j}}{2} \|x_{i,j}^{t+1} - \tilde{z}_j\|^2 && (6.64) \\
\leq & \sum_{j \in \mathcal{N}(i)} \rho_i \|\tilde{z}_j^t - z_j^t\|^2 - \left(\frac{\rho_i - 12L_{i,j}}{4}\right) \|x_{i,j}^{t+1} - x_{i,j}^t\|^2 \\
& + \sum_{j \in \mathcal{N}(i)} \frac{7L_{i,j}}{2\rho_i^2} \|y_{i,j}^{t+1} - y_{i,j}^t\|^2 && (6.65) \\
\leq & \sum_{j \in \mathcal{N}(i)} \left[- \left(\frac{\rho_i - 12L_{i,j}}{4}\right) \|x_{i,j}^{t+1} - x_{i,j}^t\|^2 + \rho_i T \sum_{t'=0}^{T-1} \|z_j^{t+1-t'} - z_j^{t-t'}\|^2 \right] \\
& + \sum_{j \in \mathcal{N}(i)} \frac{7L_{i,j}}{2\rho_i^2} \|y_{i,j}^{t+1} - y_{i,j}^t\|^2. && (6.66)
\end{aligned}
$$

The desired result then follows. □

Lemma 6.7.4. *When worker i is updating, we have*

$$
\begin{aligned}
& l_i(\mathbf{x}_i^{t+1}, \mathbf{z}^{t+1}, \mathbf{y}_i^{t+1}) - l_i(\mathbf{x}_i^{t+1}, \mathbf{z}^{t+1}, \mathbf{y}_i^t) \\
& \leq \sum_{j \in \mathcal{N}(i)} \left(\frac{1}{\rho_i} + \frac{1}{2}\right) \|y_{i,j}^{t+1} - y_{i,j}^t\|^2 + \sum_{j \in \mathcal{N}(i)} \frac{T}{2} \sum_{t'=0}^{T-1} \|\tilde{z}_{i,j}^{t+1} - z_{i,j}^{t+1}\|^2.
\end{aligned}
$$

Proof. From 6.12 we have

$$
\begin{aligned}
& l_i(\mathbf{x}_i^{t+1}, \mathbf{z}^{t+1}, \mathbf{y}_i^{t+1}) - l_i(\mathbf{x}_i^{t+1}, \mathbf{z}^{t+1}, \mathbf{y}_i^t) && (6.67) \\
= & \sum_{j \in \mathcal{N}(i)} \langle y_{i,j}^{t+1} - y_{i,j}^t, x_{i,j}^{t+1} - z_{i,j}^{t+1} \rangle && (6.68) \\
= & \sum_{j \in \mathcal{N}(i)} \langle y_{i,j}^{t+1} - y_{i,j}^t, x_{i,j}^{t+1} - \tilde{z}_{i,j}^{t+1} \rangle + \sum_{j \in \mathcal{N}(i)} \langle y_{i,j}^{t+1} - y_{i,j}^t, \tilde{z}_{i,j}^{t+1} - z_{i,j}^{t+1} \rangle && (6.69) \\
\leq & \sum_{j \in \mathcal{N}(i)} \frac{1}{\rho_i} \|y_{i,j}^{t+1} - y_{i,j}^t\|^2 + \sum_{j \in \mathcal{N}(i)} \frac{1}{2} \|y_{i,j}^{t+1} - y_{i,j}^t\|^2 + \frac{1}{2} \|\tilde{z}_{i,j}^{t+1} - z_{i,j}^{t+1}\|^2 && (6.70) \\
\leq & \sum_{j \in \mathcal{N}(i)} \left(\frac{1}{\rho_i} + \frac{1}{2}\right) \|y_{i,j}^{t+1} - y_{i,j}^t\|^2 + \sum_{j \in \mathcal{N}(i)} \frac{T}{2} \sum_{t'=0}^{T-1} \|\tilde{z}_{i,j}^{t+1} - z_{i,j}^{t+1}\|^2. && (6.71)
\end{aligned}
$$

□

Lemma 6.7.5. *After updating* $\mathbf{z}^t$ *to* $\mathbf{z}^{t+1}$*, we have*

$$L(\mathbf{X}^{t+1},\mathbf{Y}^{t+1},\mathbf{z}^{t+1})-L(\mathbf{X}^{t+1},\mathbf{Y}^{t+1},\mathbf{z}^{t})\leq-\sum_{(i,j)\in\mathcal{E}}(\gamma+\rho_i)\cdot\|z_j^{t+1}-z_j^t\|^2. \quad (6.72)$$

Proof. Firstly, it is clear that $\langle y_{i,j},x_{i,j}-z_j\rangle+\rho_i\|x_{i,j}-z_j\|^2$ is a quadratic function and thus strongly convex. Then, we have:

$$\begin{aligned}&\sum_{i\in\mathcal{N}(j)}\langle y_{i,j}^{t+1},x_{i,j}^{t+1}-z_j^{t+1}\rangle+\frac{\rho_i}{2}\|x_{i,j}^{t+1}-z_j^t\|^2-\sum_{i\in\mathcal{N}(j)}\langle y_{i,j}^{t+1},x_{i,j}^{t+1}-z_j^t\rangle-\frac{\rho_i}{2}\|x_{i,j}^{t+1}-z_j^t\|^2\\&\leq\langle-y_{i,j}^{t+1}-\rho_i(x_{i,j}^{t+1}-z_j^{t+1}),z_j^{t+1}-z_j^t\rangle-\sum_{i\in\mathcal{N}(j)}\frac{\rho_i}{2}\|z_j^{t+1}-z_j^t\|^2.\end{aligned}$$

By the optimality in (6.15), we have

$$\left\langle p_j^{t+1}-\sum_{i\in\mathcal{N}(j)}y_{i,j}^{t+1}+\rho_i(z_j^{t+1}-x_{i,j}^{t+1})+\gamma(z_j^{t+1}-z_j^t),z_j^{t+1}-z_j^t\right\rangle\leq 0,$$

where $p_j^{t+1}\in\partial h_j(z_j^{t+1})$ is a subgradient. By convexity of h_j, we have

$$\begin{aligned}h_j(z_j^{t+1})-h_j(z_j^t)&\leq\langle p_j^{t+1},z_j^{t+1}-z_j^t\rangle\\&\leq\left\langle\sum_{i\in\mathcal{N}(j)}y_{i,j}^{t+1}-\rho_i(z_j^{t+1}-x_{i,j}^{t+1})-\gamma(z_j^{t+1}-z_j^t),z_j^{t+1}-z_j^t\right\rangle\end{aligned}$$

Therefore we have

$$\begin{aligned}&L(\mathbf{X}^{t+1},\mathbf{Y}^{t+1},\mathbf{z}^{t+1})+\frac{\gamma}{2}\|\mathbf{z}^{t+1}-\mathbf{z}^t\|^2-L(\mathbf{X}^t,\mathbf{Y}^t,\mathbf{z}^t)\\&=\sum_{j=1}^M\Big(h_j(z_j^{t+1})-h_j(z_j^t)\Big)\\&\quad+\sum_{j=1}^M\sum_{i\in\mathcal{N}(j)}\langle y_{i,j}^{t+1},x_{i,j}^{t+1}-x_{i,j}^t\rangle+\frac{\rho_i}{2}\Big(\|x_{i,j}^{t+1}-z_j^{t+1}\|^2-\|x_{i,j}^{t+1}-z_j^t\|^2\Big)\\&\leq\sum_{j=1}^M\left\langle-\sum_{i\in\mathcal{N}(j)}(y_{i,j}^{t+1}-\rho_i(x_{i,j}^{t+1}-z_j^{t+1})),z_j^{t+1}-z_j^t\right\rangle-\sum_{i\in\mathcal{N}(j)}\frac{\rho_i}{2}\|z_j^{t+1}-z_j^t\|^2\\&\quad+\sum_{j=1}^M\left\langle\sum_{i\in\mathcal{N}(j)}[y_{i,j}^{t+1}-\rho_i(z_j^{t+1}-x_{i,j}^{t+1})-\gamma(z_j^{t+1}-z_j^t)],z_j^{t+1}-z_j^t\right\rangle\\&=-\sum_{(i,j)\in\mathcal{E}}\frac{\gamma+2\rho_i}{2}\cdot\|z_j^{t+1}-z_j^t\|^2.\end{aligned}$$

which proves the lemma. □

Given these three lemmas, we have

$$
\begin{aligned}
&L(\mathbf{X}^{t+1}, \mathbf{Y}^{t+1}, \mathbf{z}^{t+1}) - L(\mathbf{X}^{t}, \mathbf{Y}^{t}, \mathbf{z}^{t}) \\
\leq &- \sum_{(i,j)\in\mathcal{E}} \left[-\left(\frac{\rho_i - 12L_{i,j}}{4}\right) \|x_{i,j}^{t+1} - x_{i,j}^{t}\|^2 + \left(\rho_i + \frac{1}{2}\right) T \sum_{t'=0}^{T-1} \|z_j^{t+1-t'} - z_j^{t-t'}\|^2 \right] \\
&+ \sum_{(i,j)\in\mathcal{E}} \left(\frac{7L_{i,j}}{2\rho_i^2} + \frac{1}{\rho_i} + \frac{1}{2}\right) \|y_{i,j}^{t+1} - y_{i,j}^{t}\|^2 + (\gamma + \rho_i)\|z_j^{t+1} - z_j^{t}\|^2.
\end{aligned}
$$

From Lemma 6.7.1 we have

$$
\begin{aligned}
&L(\mathbf{X}^{t+1}, \mathbf{Y}^{t+1}, \mathbf{z}^{t+1}) - L(\mathbf{X}^{t}, \mathbf{Y}^{t}, \mathbf{z}^{t}) \\
\leq &- \sum_{(i,j)\in\mathcal{E}} \left[-\left(\frac{\rho_i - 12L_{i,j}}{4}\right) \|x_{i,j}^{t+1} - x_{i,j}^{t}\|^2 + \left(\rho_i + \frac{1}{2}\right) T \sum_{t'=0}^{T-1} \|z_j^{t+1-t'} - z_j^{t-t'}\|^2 \right] \\
&+ \sum_{(i,j)\in\mathcal{E}} \left(\frac{7L_{i,j}}{2\rho_i^2} + \frac{1}{\rho_i} + \frac{1}{2}\right) L_{i,j}^2 (T+1) \sum_{t'=0}^{T} \|z_j^{t+1-t'} - z_j^{t-t'}\|^2 + (\gamma + \rho_i)\|z_j^{t+1} - z_j^{t}\|^2.
\end{aligned}
$$

Then for any t, we take the telescope sum and we have

$$
\begin{aligned}
&L(\mathbf{X}^{t+1}, \mathbf{Y}^{t+1}, \mathbf{z}^{t+1}) - L(\mathbf{X}^{1}, \mathbf{Y}^{1}, \mathbf{z}^{1}) && (6.73) \\
\leq &- \sum_{d=1}^{t} \sum_{(i,j)\in\mathcal{E}} \beta_{i,j} \|x_{i,j}^{t+1} - x_{i,j}^{t}\|^2 - \sum_{d=1}^{t} \sum_{(i,j)\in\mathcal{E}} \alpha_{i,j} \|z_j^{t+1} - z_j^{t}\|^2, && (6.74)
\end{aligned}
$$

where

$$
\begin{aligned}
\alpha_{i,j} &:= (\gamma + \rho_i) - \left(\frac{7L_{i,j}}{2\rho_i^2} + \frac{1}{\rho_i} + \frac{1}{2}\right) L_{i,j}^2 (T+1)^2 - \left(\frac{(2+\rho_i)T^2}{2}\right), \\
\beta_{i,j} &:= \frac{\rho_i}{4} - 3L_{i,j},
\end{aligned}
$$

which proves Lemma 6.7.1.

6.7.2 Proof of Lemma 6.7.2

Proof. From Lipschitz continuity assumption, we have.

$$
\begin{aligned}
f_i(\mathbf{z}^{t+1}) &\leq f_i(\mathbf{x}_i^{t+1}) + \sum_{j\in\mathcal{N}(i)} \langle \nabla_j f_i(\mathbf{x}_i^{t+1}), z_j^{t+1} - x_{i,j}^{t+1} \rangle + \sum_{j\in\mathcal{N}(i)} \frac{L_{i,j}}{2} \|x_{i,j}^{t+1} - z_j^{t+1}\|^2 \\
&= f_i(\mathbf{x}_i^{t+1}) + \sum_{j\in\mathcal{N}(i)} \langle \nabla_j f_i(\mathbf{x}_i^{t+1}) - \nabla_j f_i(\mathbf{z}^{t+1}), z_j^{t+1} - x_{i,j}^{t+1} \rangle \\
&\quad + \sum_{j\in\mathcal{N}(i)} \langle \nabla_j f_i(\mathbf{z}^{t+1}), z_j^{t+1} - x_{i,j}^{t+1} \rangle + \sum_{j\in\mathcal{N}(i)} \frac{L_{i,j}}{2} \|x_{i,j}^{t+1} - z_j^{t+1}\|^2 \\
&\leq f_i(\mathbf{x}_i^{t+1}) + \sum_{j\in\mathcal{N}(i)} \langle \nabla_j f_i(\mathbf{z}^{t+1}), z_j^{t+1} - x_{i,j}^{t+1} \rangle + \sum_{j\in\mathcal{N}(i)} \frac{3L_{i,j}}{2} \|x_{i,j}^{t+1} - z_j^{t+1}\|^2
\end{aligned}
$$

Now we have

$$
\begin{aligned}
&L(\mathbf{X}^{t+1}, \mathbf{Y}^{t+1}, \mathbf{z}^{t+1}) \\
=&h(\mathbf{z}^{t+1}) + \sum_{i=1}^{N} f_i(\mathbf{x}_i^{t+1}) + \sum_{j\in\mathcal{N}(i)} \langle y_{i,j}^{t+1}, x_{i,j}^{t+1} - z_j^{t+1}\rangle + \frac{\rho_i}{2}\|x_{i,j}^{t+1} - z_j^{t+1}\|^2 \\
\geq&h(\mathbf{z}^{t+1}) + \sum_{i=1}^{N} f_i(\mathbf{z}^{t+1}) + \sum_{(i,j)\in\mathcal{E}} \langle \nabla_j f_i(\tilde{\mathbf{z}}^{t+1}) - \nabla_j f_i(\mathbf{z}^{t+1}), z_j^{t+1} - x_{i,j}^{t+1}\rangle \\
&+ \sum_{(i,j)\in\mathcal{E}} \frac{\rho_i - 3L_{i,j}}{2}\|x_{i,j}^{t+1} - z_j^{t+1}\|^2 \\
\geq&h(\mathbf{z}^{t+1}) + \sum_{i=1}^{N} f_i(\mathbf{z}^{t+1}) + \sum_{(i,j)\in\mathcal{E}} \frac{\rho_i - 3L_{i,j}}{2}\|x_{i,j}^{t+1} - z_j^{t+1}\|^2 \\
&- \sum_{(i,j)\in\mathcal{E}} L_{i,j}\|\tilde{\mathbf{z}}^{t+1} - \mathbf{z}^{t+1}\|\|\mathbf{z}^{t+1} - \mathbf{x}_i^{t+1}\| \\
\geq&h(\mathbf{z}^{t+1}) + \sum_{i=1}^{N} f_i(\mathbf{z}^{t+1}) + \sum_{(i,j)\in\mathcal{E}} \left(\frac{\rho_i - 4L_{i,j}}{2}\|x_{i,j}^{t+1} - z_j^{t+1}\|^2 - \frac{L_{i,j}}{2}\|\tilde{\mathbf{z}}^{t+1} - \mathbf{z}^{t+1}\|^2\right) \\
\geq&h(\mathbf{z}^{t+1}) + \sum_{i=1}^{N} f_i(\mathbf{z}^{t+1}) - \sum_{(i,j)\in\mathcal{E}} \frac{L_{i,j}}{2}\|\tilde{\mathbf{z}}^{t+1} - \mathbf{z}^{t+1}\|^2 \\
\geq&\underline{f} - \mathrm{diam}^2(\mathcal{X}) \sum_{(i,j)\in\mathcal{E}} \frac{L_{i,j}}{2} > -\infty.
\end{aligned}
$$

□

Chapter 7

Asynchronous Proximal Stochastic Gradient Descent

7.1 Background

In the previous chapter, we have seen the power of parallel algorithms and the essence of parallel optimization for large scale machine learning. Another trend to deal with large volumes of data is the use of *stochastic* algorithms. As the number of training samples n increases, the cost of updating the model $\mathbf{x}$ taking into account all error gradients becomes prohibitive. To tackle this issue, stochastic algorithms make it possible to update $\mathbf{x}$ using only a small subset of all training samples at a time. In this chapter, we consider the following *stochastic* optimization problem:

$$\min_{\mathbf{x}} \quad \Psi(\mathbf{x}) := \mathbb{E}_{\xi}[F(\mathbf{x};\xi)] + h(\mathbf{x}), \tag{7.1}$$

where the stochastic nature comes from the random variable ξ, which in our problem settings, represents a random index selected from the training set $\{1,\ldots,n\}$. Therefore, (7.1) attempts to minimize the expected loss of a random training sample plus a regularizer $h(\mathbf{x})$. In this work, we assume the function h is proper, closed and convex, yet *not necessarily smooth*.

Stochastic gradient descent (SGD) is one of the first algorithms widely implemented in an asynchronous parallel fashion; its convergence rates and speedup properties have been analyzed for both convex [30, 76] and nonconvex [75] optimization problems. Nevertheless, SGD is mainly applicable to the case of smooth optimization, and yet is not suitable for problems with a *nonsmooth* term in the objective function, e.g., an ℓ_1 norm regularizer. In fact, such nonsmooth regularizers are commonplace in many practical machine learning problems or constrained optimization

problems. In these cases, SGD becomes ineffective, as it is hard to obtain gradients for a nonsmooth objective function.

Many classical deterministic (non-stochastic) algorithms are available to solve the problem (1.1), including the proximal gradient (ProxGD) method [21] and its accelerated variants [112] as well as the alternating direction method of multipliers (ADMM) [111]. These methods leverage the so-called *proximal operators* [21] to handle the nonsmoothness in the problem. Although implementing these deterministic algorithms in a *synchronous* parallel fashion is straightforward, extending them to asynchronous parallel algorithms is much more complicated than it appears. In fact, existing theory on the convergence of asynchronous proximal gradient (PG) methods for nonconvex problem (1.1) is quite limited. An asynchronous parallel proximal gradient method has been presented in [84] and has been shown to converge to stationary points for nonconvex problems. However, [84] has essentially proposed a non-stochastic algorithm and has not provided its convergence rate.

In this chapter, we propose and analyze an asynchronous parallel *proximal stochastic gradient descent* (ProxSGD) method for solving the nonconvex and nonsmooth problem (1.1), with provable convergence and speedup guarantees. The analysis of ProxSGD has attracted much attention in the community recently. Under the assumption of an *increasing* minibatch size used in the stochastic algorithm, the non-asymptotic convergence of ProxSGD to stationary points has been shown in [68] for problem (1.1) with a convergence rate of $O(1/\sqrt{K})$, K being the times the model is updated. Moreover, additional variance reduction techniques have been introduced [25] to guarantee the convergence of ProxSGD, which is different from the stochastic method we discuss here. The stochastic algorithm considered in this chapter assumes that each worker selects a minibatch of randomly chosen training samples to calculate the gradients at a time, which is a scheme widely used in practice. To the best of our knowledge, the convergence behavior of ProxSGD—under a *constant* minibatch size without variance reduction—is still unknown (even for the synchronous or sequential version).

Notations: We introduce some additional notations for this chapter and Chapter 8 only. We use n to denote the number of *total samples*, and N to denote the mini-batch size per iteration. We use $g(\mathbf{x}) := \nabla f(\mathbf{x})$ to denote the "true" gradient, and use $G(\mathbf{x};\xi)$ to denote the stochastic gradient $\nabla F(\mathbf{x};\xi)$ w.r.t. the random variable ξ. For a random variable or vector X, let $\mathbb{E}[X|\mathcal{F}]$ be the conditional expectation

of X w.r.t. a sigma algebra $\mathcal{F}$.

The results in this chapter have appeared as technical report [6].

Algorithm 8 Asyn-ProxSGD: Asynchronous Proximal Stochastic Gradient Descent

Server executes:

Initialize $\mathbf{x}^0$.
Initialize $G \leftarrow 0$. ▷ Gradient accumulator
Initialize $s \leftarrow 0$. ▷ Request counter
loop
  if `Pull Request` from worker j is received: **then**
    Send x to worker j.
  end if
  if `Push Request` (gradient G_j) from worker j is received: **then**
    $s \leftarrow s + 1$.
    $G \leftarrow G + \frac{1}{N} \cdot G_j$.
    if $s = m$ **then**
      $\mathbf{x} \leftarrow \mathbf{prox}_{\eta h}(\mathbf{x} - \eta G)$.
      $s \leftarrow 0$.
      $G \leftarrow 0$.
    end if
  end if
end loop

Worker j asynchronously performs:

Pull x^0 to initialize.
for $t = 0, 1, \ldots$ **do**
  Randomly choose N/m training samples indexed by $\xi_{t,1}(j), \ldots, \xi_{t,N/m}(j)$.
  Calculate $G_j^t = \sum_{i=1}^{N} \nabla F(\mathbf{x}^t; \xi_{t,i}(j))$.
  Push G_j^t to the server.
  Pull the current model $\mathbf{x}$ from the server: $\mathbf{x}^{t+1} \leftarrow \mathbf{x}$.
end for

7.2 Asynchronous Proximal Gradient Descent

We now present our *asynchronous proximal gradient descent* (Asyn-ProxSGD) algorithm, which is the main contribution in this chapter. In the asynchronous algorithm, different workers may be in different local iterations due to random delays in computation and communication.

For ease of presentation, let us first assume each worker uses only one random sample at a time to compute its stochastic gradient, which naturally generalizes to using a mini-batch of random samples to compute a stochastic gradient. In this case, each worker will independently and asynchronously repeat the following steps:

- Pull the latest model x from the server;

- Calculate a gradient $\tilde{G}(\mathbf{x};\xi)$ based on a random sample ξ locally;
- Push the gradient $\tilde{G}(\mathbf{x};\xi)$ to the server.

Here we use $\tilde{G}$ to emphasize that the gradient computed on workers *may be delayed*. For example, all workers but worker j have completed their tasks of iteration t, while worker j still works on iteration $t-1$. In this case, the gradient $\tilde{G}$ is not computed based on the current model $\mathbf{x}^t$ but from a delayed one $\mathbf{x}^{t-1}$.

In our algorithm, the server will perform an averaging over the received sample gradients as long as N gradients are received and perform an proximal gradient descent update on the model $\mathbf{x}$, no matter where these N gradients come from; as long as N gradients are received, the averaging is performed. This means that it is possible that the server may have received multiple gradients from one worker while not receiving any from another worker.

In general, when each mini-batch has N samples, and each worker processes N/m random samples to calculate a stochastic gradient to be pushed to the server, the proposed Asyn-ProxSGD algorithm is described in Algorithm 8 leveraging a parameter server architecture. The server maintains a counter s. Once s reaches m, the server has received gradients that contain information about N random samples (no matter where they come from) and will perform a proximal model update.

7.3 Convergence Analysis

7.3.1 Assumptions and Metrics

We make the following assumptions for convergence analysis. We assume that $f(\cdot)$ is a smooth function with the following properties:

Assumption 7.1 (Lipschitz Gradient)**.** *For function f there are Lipschitz constants $L > 0$ such that*

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

For nonsmooth regularizer term $h(\mathbf{x})$, we assume it is a proper, closed and convex yet not necessarily smooth function as same as we assumed throughout this thesis. If the algorithm has been executed for k iterations, we let $\mathcal{F}_k$ denote the set that consists of all the samples used up to iteration k. Since $\mathcal{F}_k \subseteq \mathcal{F}_{k'}$ for all

$k \leq k'$, the collection of all such $\mathcal{F}_k$ forms a *filtration*. Under such settings, we can restrict our attention to those stochastic gradients with an unbiased estimate and bounded variance, which are common in the analysis of *stochastic* gradient descent or *stochastic* proximal gradient algorithms, e.g., [75, 68].

Assumption 7.2 (Unbiased gradient)**.** *For any k, we have $\mathbb{E}[G_k|\mathcal{F}_k] = g_k$, i.e., all stochastic gradients G_k are assumed to be an unbiased estimation of the true gradient.*

Assumption 7.3 (Bounded variance)**.** *The variance of the stochastic gradient is bounded by $\mathbb{E}[\|G(\mathbf{x};\xi) - \nabla f(\mathbf{x})\|^2] \leq \sigma^2$.*

Assumption 7.4 (Bounded delay)**.** *All delay variables $\tau(k,i)$ are bounded by T: $\max_{k,i} \tau(k,i) \leq T$.*

Assumption 7.5 (Sample Independence)**.** *All random variables including samples $\{\xi_{k,i}\}$ for all k and i in Algorithm 9 are mutually independent.*

7.3.2 Convergence Analysis for Asyn-ProxSGD

To facilitate the analysis of Algorithm 8, we rewrite it in an equivalent global view (from the server's perspective), as described in Algorithm 9. In this algorithm, we use an iteration counter k to keep track of how many times the model x has been updated on the server; k increments every time a push request (model update request) is completed. Note that such a counter k is *not* required by workers to compute gradients and is different from the counter t in Algorithm 8—t is maintained by each worker to count how many sample gradients have been computed locally.

In particular, for every N stochastic sample gradients received, the server simply aggregates them by averaging:

$$\tilde{G}^k := \frac{1}{N} \sum_{i=1}^{N} \nabla F(\mathbf{x}^{k-\tau(k,i)}; \xi_{k,i}), \tag{7.2}$$

where $\tau(k,i)$ indicates that the stochastic gradient $\nabla F(\mathbf{x}^{k-\tau(k,i)}; \xi_{k,i})$ received at iteration k could have been computed based on an older model $\mathbf{x}^{k-\tau(k,i)}$ due to communication delay and asynchrony among workers. Then, the server updates $\mathbf{x}^k$ to $\mathbf{x}^{k+1}$ using proximal gradient descent.

We present our main convergence theorem as follows:

Algorithm 9 Asyn-ProxSGD (from a Global Perspective)

1: Initialize x^1.
2: **for** $k = 1, \ldots, K$ **do**
3: Randomly select N training samples indexed by $\xi_{k,1}, \ldots, \xi_{k,N}$.
4: Calculate the averaged gradient $\tilde{G}^k$ according to (7.2).
5: $\mathbf{x}^{k+1} \leftarrow \mathbf{prox}_{\eta_k h}(\mathbf{x}^k - \eta_k \tilde{G}^k)$.
6: **end for**

Theorem 7.3.1. *If Assumptions 7.4 and 7.5 hold and the step length sequence* $\{\eta_k\}$ *in Algorithm 9 satisfies*

$$\eta_k \leq \frac{1}{16L}, \quad 6\eta_k L^2 T \sum_{l=1}^{T} \eta_{k+l} \leq 1, \tag{7.3}$$

for all $k = 1, 2, \ldots, K$*, we have the following ergodic convergence rate for Algorithm 9:*

$$\begin{aligned} &\frac{\sum_{k=1}^{K}(\eta_k - 8L\eta_k^2)\mathbb{E}[\|\mathcal{P}(\mathbf{x}^k)\|^2]}{\sum_{k=1}^{K}(\eta_k - 8L\eta_k^2)} \\ \leq\ &\frac{8(\Psi(\mathbf{x}^1) - \Psi(\mathbf{x}^*))}{\sum_{k=1}^{K}\eta_k - 8L\eta_k^2} + \frac{\sum_{k=1}^{K}\left(8L\eta_k^2 + 12\eta_k L^2 T\sum_{l=1}^{T}\eta_{k-l}^2\right)\sigma^2}{N\sum_{k=1}^{K}(\eta_k - 8L\eta_k^2)}, \end{aligned} \tag{7.4}$$

where the expectation is taken in terms of all random variables in Algorithm 9.

Taking a closer look at Theorem 7.3.1, we can properly choose the learning rate η_k as a constant value and derive the following convergence rate:

Corollary 7.3.1. *Let the step length be a constant, i.e.,*

$$\eta := \sqrt{\frac{(\Psi(\mathbf{x}^1) - \Psi(\mathbf{x}^*))N}{2KL\sigma^2}}. \tag{7.5}$$

If the delay bound T *satisfies*

$$K \geq \frac{128(\Psi(\mathbf{x}^1) - \Psi(\mathbf{x}^*))NL}{\sigma^2}(T+1)^4, \tag{7.6}$$

then the output of Algorithm 7.3.1 satisfies the following ergodic convergence rate:

$$\min_{k=1,\ldots,K}\mathbb{E}[\|\mathcal{P}(\mathbf{x}^k)\|^2] \leq \frac{1}{K}\sum_{k=1}^{K}\mathbb{E}[\|\mathcal{P}(\mathbf{x}^k)\|^2] \leq 32\sqrt{\frac{2\left(\Psi(\mathbf{x}^1) - \Psi(\mathbf{x}^*)\right)L\sigma^2}{KN}}. \tag{7.7}$$

Remark 1. (Consistency with ProxSGD) When $T = 0$, our proposed Asyn-ProxSGD reduces to the vanilla ProxSGD (e.g., [68]). Thus, the iteration complexity

is $O(1/\epsilon^2)$ according to (7.7), attaining the same result as that in [68] *yet without assuming increased mini-batch sizes.*

Remark 2. (Linear speedup w.r.t. the staleness) From (7.7) we can see that linear speedup is achievable, as long as the delay T is bounded by $O(K^{1/4})$ (if other parameters are constants). The reason is that by (7.6) and (7.7), as long as T is no more than $O(K^{1/4})$, the iteration complexity (from a global perspective) to achieve ϵ-optimality is $O(1/\epsilon^2)$, which is independent from T.

Remark 3. (Linear speedup w.r.t. number of workers) As the iteration complexity is $O(1/\epsilon^2)$ to achieve ϵ-optimality, it is also independent from the number of workers m if assuming other parameters are constants. It is worth noting that the delay bound T is roughly proportional to the number of workers. As the iteration complexity is independent from T, we can conclude that the total iterations will be shortened to $1/T$ of a single worker's iterations if $\Theta(T)$ workers work in parallel, achieving nearly linear speedup.

Remark 4. (Comparison with Asyn-SGD) Compared with asynchronous SGD [75], in which T or the number of workers should be bounded by $O(\sqrt{K/N})$ to achieve linear speedup, here Asyn-ProxSGD is more sensitive to delays and more suitable for a smaller cluster.

7.4 Experiments

We now present experimental results to confirm the capability and efficiency of our proposed algorithm to solve challenging nonconvex nonsmooth machine learning problems. We implemented our algorithm on TensorFlow [18], a flexible and efficient deep learning library. We execute our algorithm on `Ray` [113], a general-purpose framework that enables parallel and distributed execution of Python as well as TensorFlow functions. A key feature of `Ray` is that it provides a unified task-parallel abstraction, which can serve as workers, and actor abstraction, which stores some states and acts like parameter servers.

We use a cluster of 9 instances on Google Cloud. Each instance has one CPU core with 3.75 GB RAM, running 64-bit Ubuntu 16.04 LTS. Each server or worker uses only one core, with 9 CPU cores and 60 GB RAM used in total. Only one instance is the server node, while the other nodes are workers.

Setup: In our experiments, we consider the problem of non-negative principle component analysis (NN-PCA) [25]. Given a set of n samples $\{\mathbf{z}_i\}_{i=1}^n$, NN-PCA

Table 7.1: Iteration speedup and time speedup of Asyn-ProxSGD at the suboptimality level 10^{-3}. (a9a)

Workers	1	2	4	8
Iteration Speedup	1.000	1.982	3.584	5.973
Time Speedup	1.000	2.219	3.857	5.876

Table 7.2: Description of the two classification datasets used.

datasets	dimension	sample size
a9a	123	32,561
mnist	780	60,000

solves the following optimization problem

$$\min_{\|\mathbf{x}\|\leq 1, \mathbf{x}\geq 0} -\frac{1}{2}\mathbf{x}^\top \left(\sum_{i=1}^{n} \mathbf{z}_i \mathbf{z}_i^\top\right) \mathbf{x}. \tag{7.8}$$

This NN-PCA problem is NP-hard in general. To apply our algorithm, we can rewrite it with $f_i(\mathbf{x}) = -(\mathbf{x}^\top \mathbf{z}_i)^2/2$ for all samples $i \in [n]$. Since the feasible set $C = \{\mathbf{x} \in \mathbb{R}^d : \|\mathbf{x}\| \leq 1, \mathbf{x} \geq 0\}$ is convex, we can replace the optimization constraint by a regularizer in the form of an indicator function $h(\mathbf{x}) = I_C(\mathbf{x})$, such that $h(\mathbf{x}) = 0$ if $\mathbf{x} \in C$ and ∞ otherwise.

The hyper-parameters are set as follows. The step size is set using the popular t-inverse step size choice $\eta_k = \eta_0/(1 + \eta'(k/k'))$, which is the same as the one used in [25]. Here $\eta_0, \eta' > 0$ determine how learning rates change, and k' controls for how many steps the learning rate would change.

We conduct experiments on two datasets [1], with their information summarized in Table 7.2. All samples have been normalized, i.e., $\|\mathbf{z}_i\| = 1$ for all $i \in [n]$. In our experiments, we use a batch size of $N = 8192$ in order to evaluate the performance and speedup behavior of the algorithm under constant batches.

We consider the *function suboptimality* value as our performance metric. In particular, we run proximal gradient descent (ProxGD) for a large number of iterations with multiple random initializations, and obtain a solution $\hat{\mathbf{x}}$. For all experiments, we evaluate function suboptimality, which is the gap $f(\mathbf{x}) - f(\hat{\mathbf{x}})$, against the number of sample gradients processed by the server (divided by the total number of samples n), and then against time.

Results: Empirically, Assumption 7.4 (bounded delays) is observed to hold for this cluster. For our proposed Asyn-ProxSGD algorithm, we are particularly

[1] Available at http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/

interested in the speedup in terms of iterations and running time. In particular, if we need T_1 iterations (with T_1 sample gradients processed by the server) to achieve a certain suboptimality level using one worker, and T_p iterations (with T_p sample gradients processed by the server) to achieve the same suboptimality with p workers, the iteration speedup is defined as $p \times T_1/T_p$ [75]. Note that all iterations are counted on the server side, i.e., how many sample gradients are processed by the server. On the other hand, the running time speedup is defined as the ratio between the running time of using one worker and that of using p workers to achieve the same suboptimality.

The iteration and running time speedups on both datasets are shown in Fig. 7.1 and Fig. 7.2, respectively. Such speedups achieved at the suboptimality level of 10^{-3} are presented in Table 7.1 and 7.3. We observe that nearly linear speedup can be achieved, although there is a loss of efficiency due to communication as the number workers increases.

Table 7.3: Iteration speedup and time speedup of Asyn-ProxSGD at the suboptimality level 10^{-3}. (`mnist`)

Workers	1	2	4	8
Iteration Speedup	1.000	2.031	3.783	7.352
Time Speedup	1.000	2.285	4.103	5.714

7.5 Proof of Theorem 7.3.1

Our proof of Theorem 7.3.1 relies on the following milestone lemmas. In this proof, we denote $\delta^k := \tilde{g}^k - \tilde{G}^k$ as the difference between these two differences.

7.5.1 Milestone lemmas

We put some key results of convergence analysis as milestone lemmas listed below, and the detailed proof is listed in 7.7.

Lemma 7.5.1 (Decent Lemma)**.**

$$\mathbb{E}[\Psi(\mathbf{x}^{k+1}) \leq \mathbb{E}[\Psi(\mathbf{x}^k)|\mathcal{F}_k] - \frac{\eta_k - 4L\eta_k^2}{2}\|P(\mathbf{x}^k, \tilde{g}^k, \eta_k)\|^2 + \frac{\eta_k}{2}\|g^k - \tilde{g}^k\|^2 + \frac{L\eta_k^2}{N}\sigma^2. \tag{7.9}$$

Lemma 7.5.2. *Suppose we have a sequence $\{\mathbf{x}^k\}$ by Algorithm 9, then we have:*

$$\mathbb{E}[\|\mathbf{x}^k - \mathbf{x}^{k-\tau}\|^2] \leq \left(\frac{2\tau}{N}\sum_{l=1}^{\tau}\eta_{k-l}^2\right)\sigma^2 + 2\left\|\sum_{l=1}^{\tau}\eta_{k-l}P(\mathbf{x}^{k-l}, \tilde{g}^{k-l}, \eta_{k-l})\right\|^2. \quad (7.10)$$

for all $\tau > 0$.

Lemma 7.5.3. *Suppose we have a sequence $\{\mathbf{x}^k\}$ by Algorithm 9, , then we have:*

$$\mathbb{E}[\|g^k - \tilde{g}^k\|^2] \leq \left(\frac{2L^2T}{N}\sum_{l=1}^{T}\eta_{k-l}^2\right)\sigma^2 + 2L^2T\sum_{l=1}^{T}\eta_{k-l}^2\|P(\mathbf{x}^{k-l}, \tilde{g}^{k-l}, \eta_{k-l})\|^2. \quad (7.11)$$

We now proceed to prove Theorem 7.3.1.

Proof. From the fact $2\|a\|^2 + 2\|b\|^2 \geq \|a+b\|^2$, we have

$$\begin{aligned}\|P(\mathbf{x}^k, \tilde{g}^k, \eta_k)\|^2 + \|g^k - \tilde{g}^k\|^2 &\geq \|P(\mathbf{x}^k, \tilde{g}^k, \eta_k)\|^2 + \|P(\mathbf{x}^k, g^k, \eta_k) - P(\mathbf{x}^k, \tilde{g}^k, \eta_k)\|^2 \\ &\geq \frac{1}{2}\|P(\mathbf{x}^k, g^k, \eta_k)\|^2,\end{aligned}$$

which implies that

$$\|P(\mathbf{x}^k, \tilde{g}^k, \eta_k)\|^2 \geq \frac{1}{2}\|P(\mathbf{x}^k, g^k, \eta_k)\|^2 - \|g^k - \tilde{g}^k\|^2.$$

We start the proof from Lemma 7.5.1. According to our condition of $\eta \leq \frac{1}{16L}$, we have $8L\eta_k^2 - \eta < 0$ and therefore

$$\begin{aligned}&\mathbb{E}[\Psi(\mathbf{x}^{k+1})|\mathcal{F}_k] \\ &\leq \mathbb{E}[\Psi(\mathbf{x}^k)|\mathcal{F}_k] + \frac{\eta_k}{2}\|g^k - \tilde{g}^k\|^2 + \frac{4L\eta_k^2 - \eta_k}{2}\|P(\mathbf{x}^k, \tilde{g}^k, \eta_k)\|^2 + \frac{L\eta_k^2}{N}\sigma^2 \\ &= \mathbb{E}[\Psi(\mathbf{x}^k)|\mathcal{F}_k] + \frac{\eta_k}{2}\|g^k - \tilde{g}^k\|^2 + \frac{8L\eta_k^2 - \eta_k}{4}\|P(\mathbf{x}^k, \tilde{g}^k, \eta_k)\|^2 - \frac{\eta_k}{4}\|P(\mathbf{x}^k, \tilde{g}^k, \eta_k)\|^2 + \frac{L\eta_k^2}{N}\sigma^2 \\ &\leq \mathbb{E}[\Psi(\mathbf{x}^k)|\mathcal{F}_k] + \frac{\eta_k}{2}\|g^k - \tilde{g}^k\|^2 + \frac{L\eta_k^2}{N}\sigma^2 + \frac{8L\eta_k^2 - \eta_k}{4}(\frac{1}{2}\|P(\mathbf{x}^k, g^k, \eta_k)\|^2 - \|g^k - \tilde{g}^k\|^2) \\ &\quad - \frac{\eta_k}{4}\|P(\mathbf{x}^k, \tilde{g}^k, \eta_k)\|^2 \\ &\leq \mathbb{E}[\Psi(\mathbf{x}^k)|\mathcal{F}_k] - \frac{\eta_k - 8L\eta_k^2}{8}\|P(\mathbf{x}^k, g^k, \eta_k)\|^2 + \frac{3\eta_k}{4}\|g^k - \tilde{g}^k\|^2 - \frac{\eta_k}{4}\|P(\mathbf{x}^k, \tilde{g}^k, \eta_k)\|^2 + \frac{L\eta_k^2}{N}\sigma^2.\end{aligned}$$

Apply Lemma 7.5.3 we have

$$\begin{aligned}&\mathbb{E}[\Psi(\mathbf{x}^{k+1})|\mathcal{F}_k] \\ &\leq \mathbb{E}[\Psi(\mathbf{x}^k)|\mathcal{F}_k] - \frac{\eta_k - 8L\eta_k^2}{8}\|P(\mathbf{x}^k, g^k, \eta_k)\|^2 + \frac{L\eta_k^2}{N}\sigma^2 - \frac{\eta_k}{4}\|P(\mathbf{x}^k, \tilde{g}^k, \eta_k)\|^2 \\ &\quad + \frac{3\eta_k}{4}\left(\frac{2L^2T}{N}\sum_{l=1}^{T}\eta_{k-l}^2\sigma^2 + 2L^2T\sum_{l=1}^{T}\eta_{k-l}^2\|P(\mathbf{x}^{k-l}, \tilde{g}^{k-l}, \eta_{k-l})\|^2\right) \\ &= \mathbb{E}[\Psi(\mathbf{x}^k)|\mathcal{F}_k] - \frac{\eta_k - 8L\eta_k^2}{8}\|P(\mathbf{x}^k, g^k, \eta_k)\|^2 + \left(\frac{L\eta_k^2}{N} + \frac{3\eta_k L^2T}{2N}\sum_{l=1}^{T}\eta_{k-l}^2\right)\sigma^2 \\ &\quad - \frac{\eta_k}{4}\|P(\mathbf{x}^k, \tilde{g}^k, \eta_k)\|^2 + \frac{3\eta_k L^2T}{2}\sum_{l=1}^{T}\eta_{k-l}^2\|P(\mathbf{x}^{k-l}, \tilde{g}^{k-l}, \eta_{k-l})\|^2.\end{aligned}$$

By taking telescope sum, we have

$$\begin{aligned}
&\mathbb{E}[\Psi(\mathbf{x}^{K+1})|\mathcal{F}_K] \\
&\leq \Psi(\mathbf{x}^1) - \sum_{k=1}^{K} \frac{\eta_k - 8L\eta_k^2}{8} \|P(\mathbf{x}^k, g^k, \eta_k)\|^2 - \sum_{k=1}^{K} \left(\frac{\eta_k}{4} - \frac{3\eta_k^2 L^2 T}{2} \sum_{l=1}^{l_k} \eta_{k+l} \right) \|P(\mathbf{x}^k, \tilde{g}^k, \eta_k)\|^2 \\
&\quad + \sum_{k=1}^{K} \left(\frac{L\eta_k^2}{N} + \frac{3\eta_k L^2 T}{2N} \sum_{l=1}^{T} \eta_{k-l}^2 \right) \sigma^2
\end{aligned}$$

where $l_k := \min(k + T - 1, K)$, and we have

$$\begin{aligned}
&\sum_{k=1}^{K} \frac{\eta_k - 8L\eta_k^2}{8} \|P(\mathbf{x}^k, g^k, \eta_k)\|^2 \\
&\leq \Psi(\mathbf{x}^1) - \mathbb{E}[\Psi(\mathbf{x}^{K+1})|\mathcal{F}_K] - \sum_{k=1}^{K} \left(\frac{\eta_k}{4} - \frac{3\eta_k^2 L^2 T}{2} \sum_{l=1}^{l_k} \eta_{k+l} \right) \|P(\mathbf{x}^k, \tilde{g}^k, \eta_k)\|^2 \\
&\quad + \sum_{k=1}^{K} \left(\frac{L\eta_k^2}{N} + \frac{3\eta_k L^2 T}{2N} \sum_{l=1}^{T} \eta_{k-l}^2 \right) \sigma^2.
\end{aligned}$$

When $6\eta_k L^2 T \sum_{l=1}^{T} \eta_{k+l} \leq 1$ for all k as the condition of Theorem 7.3.1, we have

$$\begin{aligned}
&\sum_{k=1}^{K} \frac{\eta_k - 8L\eta_k^2}{8} \|P(\mathbf{x}^k, g^k, \eta_k)\|^2 \\
&\leq \Psi(\mathbf{x}^1) - \mathbb{E}[\Psi(\mathbf{x}^{K+1})|\mathcal{F}_K] + \sum_{k=1}^{K} \left(\frac{L\eta_k^2}{N} + \frac{3\eta_k L^2 T}{2N} \sum_{l=1}^{T} \eta_{k-l}^2 \right) \sigma^2 \\
&\leq \Psi(\mathbf{x}^1) - F^* + \sum_{k=1}^{K} \left(\frac{L\eta_k^2}{N} + \frac{3\eta_k L^2 T}{2N} \sum_{l=1}^{T} \eta_{k-l}^2 \right) \sigma^2,
\end{aligned}$$

which proves the theorem. □

7.6 Proof of Corollary 7.3.1

Proof. From the condition of Corollary, we have

$$\eta \leq \frac{1}{16L(T+1)^2}.$$

It is clear that the above inequality also satisfies the condition in Theorem 7.3.1. By doing so, we can have Furthermore, we have

$$\begin{aligned}
\frac{3LT^2\eta}{2} &\leq \frac{3LT^2}{2} \cdot \frac{1}{16L(T+1)^2} \leq 1, \\
\frac{3L^2T^2\eta^3}{2} &\leq L\eta^2.
\end{aligned}$$

Since $\eta \leq \frac{1}{16L}$, we have $2 - 16L\eta^2 \geq 1$ and thus

$$\frac{8}{\eta - 8L\eta^2} = \frac{16}{\eta(2 - 16L\eta^2)} \leq \frac{16}{\eta}.$$

Following Theorem 7.3.1 and the above inequality, we have

$$\begin{aligned}
&\frac{1}{K}\sum_{k=1}^{K}\mathbb{E}[\|P(\mathbf{x}^k, g^k, \eta_k)\|^2] \\
&\leq \frac{16(\Psi(\mathbf{x}^1)-\Psi(\mathbf{x}^*))}{K\eta} + 16\left(\frac{L\eta^2}{N} + \frac{3\eta L^2 T}{2N}\sum_{l=1}^{T}\eta^2\right)\frac{K\sigma^2}{K\eta} \\
&= \frac{16(\Psi(\mathbf{x}^1)-\Psi(\mathbf{x}^*)}{K\eta} + 16\left(\frac{L\eta^2}{N} + \frac{3L^2T^2\eta^3}{2N}\right)\frac{\sigma^2}{\eta} \\
&\leq \frac{16(\Psi(\mathbf{x}^1)-\Psi(\mathbf{x}^*))}{K\eta} + \frac{32L\eta^2}{N}\cdot\frac{\sigma^2}{\eta} \\
&= \frac{16(\Psi(\mathbf{x}^1)-\Psi(\mathbf{x}^*))}{K\eta} + \frac{32L\eta\sigma^2}{N} \\
&= 32\sqrt{\frac{2(\Psi(\mathbf{x}^1)-\Psi(\mathbf{x}^*))L\sigma^2}{KN}},
\end{aligned}$$

which proves the corollary. □

7.7 Proof of Milestone Lemmas

Proof of Lemma 7.5.1. Let $\bar{\mathbf{x}}^{k+1} = \mathbf{prox}_{\eta_k h}(\mathbf{x}^k - \eta_k\tilde{g}^k)$ and apply Lemma A.1.4, we have

$$\begin{aligned}
\Psi(\mathbf{x}^{k+1}) \leq \Psi(\bar{\mathbf{x}}^{k+1}) + \langle \mathbf{x}^{k+1}-\bar{\mathbf{x}}^{k+1}, \nabla f(\mathbf{x}^k) - \tilde{G}^k\rangle + \left(\frac{L}{2} - \frac{1}{2\eta_k}\right)\|\mathbf{x}^{k+1}-\mathbf{x}^k\|^2 \\
+ \left(\frac{L}{2} + \frac{1}{2\eta_k}\right)\|\bar{\mathbf{x}}^{k+1}-\mathbf{x}^k\|^2 - \frac{1}{2\eta_k}\|\mathbf{x}^{k+1}-\bar{\mathbf{x}}^{k+1}\|^2.
\end{aligned} \tag{7.12}$$

Now we turn to bound $\Psi(\bar{\mathbf{x}}^{k+1})$ as follows:

$$\begin{aligned}
f(\bar{\mathbf{x}}^{k+1}) &\leq f(\mathbf{x}^k) + \langle \nabla f(\mathbf{x}^k), \bar{\mathbf{x}}^{k+1}-\mathbf{x}^k\rangle + \frac{L}{2}\|\bar{\mathbf{x}}^{k+1}-\mathbf{x}^k\|^2 \\
&= f(\mathbf{x}^k) + \langle g^k, \bar{\mathbf{x}}^{k+1}-\mathbf{x}^k\rangle + \frac{\eta_k^2 L}{2}\|P(\mathbf{x}^k, \tilde{g}^k, \eta_k)\|^2 \\
&= f(\mathbf{x}^k) + \langle \tilde{g}^k, \bar{\mathbf{x}}^{k+1}-\mathbf{x}^k\rangle + \langle g^k-\tilde{g}^k, \bar{\mathbf{x}}^{k+1}-\mathbf{x}^k\rangle + \frac{\eta_k^2 L}{2}\|P(\mathbf{x}^k, \tilde{g}^k, \eta_k)\|^2 \\
&= f(\mathbf{x}^k) - \eta_k\langle \tilde{g}^k, P(\mathbf{x}^k, \tilde{g}^k, \eta_k)\rangle + \langle g^k-\tilde{g}^k, \bar{\mathbf{x}}^{k+1}-\mathbf{x}^k\rangle + \frac{\eta_k^2 L}{2}\|P(\mathbf{x}^k, \tilde{g}^k, \eta_k)\|^2 \\
&\leq f(\mathbf{x}^k) - [\eta_k\|P(\mathbf{x}^k, \tilde{g}^k, \eta_k)\|^2 + h(\bar{\mathbf{x}}^{k+1}) - h(\mathbf{x}^k)] + \langle g^k-\tilde{g}^k, \bar{\mathbf{x}}^{k+1}-\mathbf{x}^k\rangle \\
&\quad + \frac{\eta_k^2 L}{2}\|P(\mathbf{x}^k, \tilde{g}^k, \eta_k)\|^2,
\end{aligned}$$

where the last inequality follows from Lemma A.1.1. By rearranging terms on both sides, we have

$$\Psi(\bar{\mathbf{x}}^{k+1}) \leq \Psi(\mathbf{x}^k) - (\eta_k - \frac{\eta_k^2 L}{2})\|P(\mathbf{x}^k, \tilde{g}^k, \eta_k)\|^2 + \langle g^k-\tilde{g}^k, \bar{\mathbf{x}}^{k+1}-\mathbf{x}^k\rangle \tag{7.13}$$

Taking the summation of (7.12) and (7.13), we have

$$
\begin{aligned}
&\Psi(\mathbf{x}^{k+1}) \\
&\leq \Psi(\mathbf{x}^k) + \langle \mathbf{x}^{k+1} - \bar{\mathbf{x}}^{k+1}, \nabla f(\mathbf{x}^k) - \tilde{G}^k \rangle \\
&\quad + \left(\frac{L}{2} - \frac{1}{2\eta_k} \right) \|\mathbf{x}^{k+1} - \mathbf{x}^k\|^2 + \left(\frac{L}{2} + \frac{1}{2\eta_k} \right) \|\bar{\mathbf{x}}^{k+1} - \mathbf{x}^k\|^2 - \frac{1}{2\eta_k} \|\mathbf{x}^{k+1} - \bar{\mathbf{x}}^{k+1}\|^2 \\
&\quad - (\eta_k - \frac{\eta_k^2 L}{2}) \|P(\mathbf{x}^k, \tilde{g}^k, \eta_k)\|^2 + \langle g^k - \tilde{g}^k, \bar{\mathbf{x}}^{k+1} - \mathbf{x}^k \rangle \\
&= \Psi(\mathbf{x}^k) + \langle \mathbf{x}^{k+1} - \mathbf{x}^k, g^k - \tilde{g}^k \rangle + \langle \mathbf{x}^{k+1} - \bar{\mathbf{x}}^{k+1}, \delta^k \rangle \\
&\quad + \left(\frac{L\eta_k^2}{2} - \frac{\eta_k}{2} \right) \|P(\mathbf{x}^k, \tilde{G}^k, \eta_k)\|^2 + \left(\frac{L\eta_k^2}{2} + \frac{\eta_k}{2} \right) \|P(\mathbf{x}^k, \tilde{g}^k, \eta_k)\|^2 \\
&\quad - \frac{1}{2\eta_k} \|\mathbf{x}^{k+1} - \bar{\mathbf{x}}^{k+1}\|^2 - (\eta_k - \frac{\eta_k^2 L}{2}) \|P(\mathbf{x}^k, \tilde{g}^k, \eta_k)\|^2 \\
&= \Psi(\mathbf{x}^k) + \langle \mathbf{x}^{k+1} - \mathbf{x}^k, g^k - \tilde{g}^k \rangle + \langle \mathbf{x}^{k+1} - \bar{\mathbf{x}}^{k+1}, \delta^k \rangle + \frac{L\eta_k^2 - \eta_k}{2} \|P(\mathbf{x}^k, \tilde{G}^k, \eta_k)\|^2 \\
&\quad + \frac{2L\eta_k^2 - \eta_k}{2} \|P(\mathbf{x}^k, \tilde{g}^k, \eta_k)\|^2 - \frac{1}{2\eta_k} \|\mathbf{x}^{k+1} - \bar{\mathbf{x}}^{k+1}\|^2
\end{aligned}
$$

By taking the expectation on condition of filtration $\mathcal{F}_k$ and according to Assumption 7.2, we have

$$
\begin{aligned}
&\mathbb{E}[\Psi(\mathbf{x}^{k+1}) | \mathcal{F}_k] \\
&\leq \mathbb{E}[\Psi(\mathbf{x}^k) | \mathcal{F}_k] + \mathbb{E}[\langle \mathbf{x}^{k+1} - \mathbf{x}^k, g^k - \tilde{g}^k \rangle | \mathcal{F}_k] + \frac{L\eta_k^2 - \eta_k}{2} \mathbb{E}[\|P(\mathbf{x}^k, \tilde{G}^k, \eta_k)\|^2 | \mathcal{F}_k] \\
&\quad + \frac{2L\eta_k^2 - \eta_k}{2} \|P(\mathbf{x}^k, \tilde{g}^k, \eta_k)\|^2 - \frac{1}{2\eta_k} \|\mathbf{x}^{k+1} - \bar{\mathbf{x}}^{k+1}\|^2.
\end{aligned} \tag{7.14}
$$

Therefore, we have

$$
\begin{aligned}
&\mathbb{E}[\Psi(\mathbf{x}^{k+1}) | \mathcal{F}_k] \\
&\leq \mathbb{E}[\Psi(\mathbf{x}^k) | \mathcal{F}_k] + \mathbb{E}[\langle \mathbf{x}^{k+1} - \mathbf{x}^k, g^k - \tilde{g}^k \rangle | \mathcal{F}_k] + \frac{L\eta_k^2 - \eta_k}{2} \mathbb{E}[\|P(\mathbf{x}^k, \tilde{G}^k, \eta_k)\|^2 | \mathcal{F}_k] \\
&\quad + \frac{2L\eta_k^2 - \eta_k}{2} \|P(\mathbf{x}^k, \tilde{g}^k, \eta_k)\|^2 - \frac{1}{2\eta_k} \|\mathbf{x}^{k+1} - \bar{\mathbf{x}}^{k+1}\|^2 \\
&\leq \mathbb{E}[\Psi(\mathbf{x}^k) | \mathcal{F}_k] + \frac{\eta_k}{2} \|g^k - \tilde{g}^k\|^2 + \frac{L\eta_k^2}{2} \mathbb{E}[\|P(\mathbf{x}^k, \tilde{G}^k, \eta_k)\|^2 | \mathcal{F}_k] + \frac{2L\eta_k^2 - \eta_k}{2} \|P(\mathbf{x}^k, \tilde{g}^k, \eta_k)\|^2 \\
&\leq \mathbb{E}[\Psi(\mathbf{x}^k) | \mathcal{F}_k] - \frac{\eta_k - 4L\eta_k^2}{2} \|P(\mathbf{x}^k, \tilde{g}^k, \eta_k)\|^2 + \frac{\eta_k}{2} \|g^k - \tilde{g}^k\|^2 + \frac{L\eta_k^2}{N} \sigma^2
\end{aligned}
$$

□

Proof of Lemma 7.5.2. Following the definition of $\mathbf{x}^k$ from Algorithm 9, we have

$$\begin{aligned}
&\|\mathbf{x}^k - \mathbf{x}^{k-\tau}\|^2 \\
&= \left\| \sum_{l=1}^{\tau} \mathbf{x}^{k-l} - \mathbf{x}^{k-l+1} \right\|^2 \\
&= \left\| \sum_{l=1}^{\tau} \eta_{k-l} P(\mathbf{x}^{k-l}, \tilde{G}^{k-l}, \eta_{k-l}) \right\|^2 \\
&= 2 \left\| \sum_{l=1}^{\tau} \eta_{k-l} [P(\mathbf{x}^{k-l}, \tilde{G}^{k-l}, \eta_{k-l}) - P(\mathbf{x}^{k-l}, \tilde{g}^{k-l}, \eta_{k-l})] \right\|^2 + 2 \left\| \sum_{l=1}^{\tau} \eta_{k-l} P(\mathbf{x}^{k-l}, \tilde{g}^{k-l}, \eta_{k-l}) \right\|^2 \\
&\leq 2\tau \sum_{l=1}^{\tau} \eta_{k-l}^2 \|P(\mathbf{x}^{k-l}, \tilde{G}^{k-l}, \eta_{k-l}) - P(\mathbf{x}^{k-l}, \tilde{g}^{k-l}, \eta_{k-l})\|^2 + 2 \left\| \sum_{l=1}^{\tau} \eta_{k-l} P(\mathbf{x}^{k-l}, \tilde{g}^{k-l}, \eta_{k-l}) \right\|^2 \\
&\leq 2\tau \sum_{l=1}^{\tau} \eta_{k-l}^2 \|\tilde{G}^{k-l} - \tilde{g}^{k-l}\|^2 + 2 \left\| \sum_{l=1}^{\tau} \eta_{k-l} P(\mathbf{x}^{k-l}, \tilde{g}^{k-l}, \eta_{k-l}) \right\|^2,
\end{aligned}$$

where the last inequality is from Lemma A.1.3. By taking the expectation on both sides, we have

$$\begin{aligned}
\mathbb{E}[\|\mathbf{x}^k - \mathbf{x}^{k-\tau}\|^2] &\leq 2\tau \sum_{l=1}^{\tau} \eta_{k-l}^2 \|\tilde{G}^{k-l} - \tilde{g}^{k-l}\|^2 + 2 \left\| \sum_{l=1}^{\tau} \eta_{k-l} P(\mathbf{x}^{k-l}, \tilde{g}^{k-l}, \eta_{k-l}) \right\|^2 \\
&\leq \frac{2\tau}{N} \sigma^2 \sum_{l=1}^{\tau} \eta_{k-l}^2 + 2 \left\| \sum_{l=1}^{\tau} \eta_{k-l} P(\mathbf{x}^{k-l}, \tilde{g}^{k-l}, \eta_{k-l}) \right\|^2,
\end{aligned}$$

which proves the lemma. □

Proof of Lemma 7.5.3. From Assumption 7.1 we have

$$\|g^k - \tilde{g}^k\|^2 = \left\| \frac{1}{N} \sum_{i=1}^{N} g^k - \tilde{g}^{t(k,i)} \right\|^2 \leq \frac{L^2}{N} \sum_{i=1}^{N} \|\mathbf{x}^k - \mathbf{x}^{k-\tau(k,i)}\|^2.$$

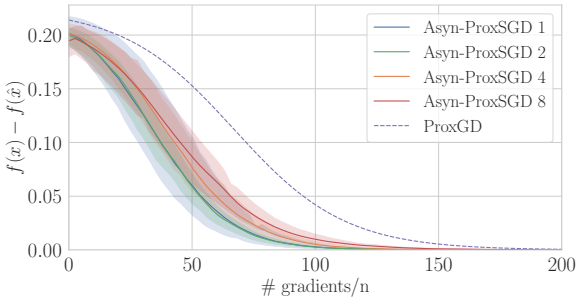
By applying Lemma 7.5.2, we have

$$\mathbb{E}[\|\mathbf{x}^k - \mathbf{x}^{k-\tau(k,i)}\|^2] \leq \frac{2\tau(k,i)}{N} \sigma^2 \sum_{l=1}^{\tau(k,i)} \eta_{k-l}^2 + 2 \left\| \sum_{l=1}^{\tau(k,i)} \eta_{k-l} P(\mathbf{x}^{k-l}, \tilde{g}^{k-l}, \eta_{k-l}) \right\|^2.$$

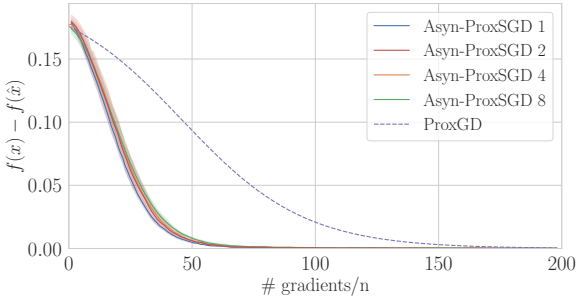
Therefore, we have

$$\begin{aligned}
\mathbb{E}[\|g^k - \tilde{g}^k\|^2] &\leq \frac{L^2}{N} \sum_{i=1}^{N} \|\mathbf{x}^k - \mathbf{x}^{k-\tau(k,i)}\|^2 \\
&\leq \frac{L^2}{N} \sum_{i=1}^{N} \left(\frac{2\tau(k,i)}{N} \sigma^2 \sum_{l=1}^{\tau(k,i)} \eta_{k-l}^2 + 2\tau(k,i) \sum_{l=1}^{\tau(k,i)} \eta_{k-l}^2 \|P(\mathbf{x}^{k-l}, \tilde{g}^{k-l}, \eta_{k-l})\|^2 \right) \\
&\leq \left(\frac{2L^2 T}{N} \sum_{l=1}^{T} \eta_{k-l}^2 \right) \sigma^2 + 2L^2 T \sum_{l=1}^{T} \eta_{k-l}^2 \|P(\mathbf{x}^{k-l}, \tilde{g}^{k-l}, \eta_{k-l})\|^2,
\end{aligned}$$

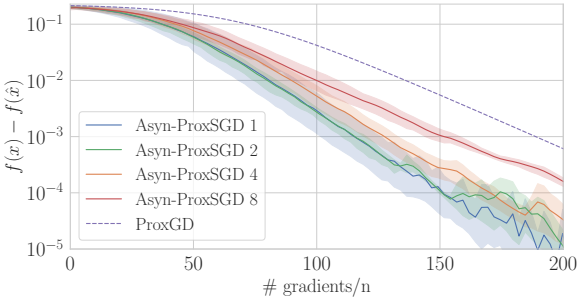
where the last inequality follows from and now we prove the lemma. □



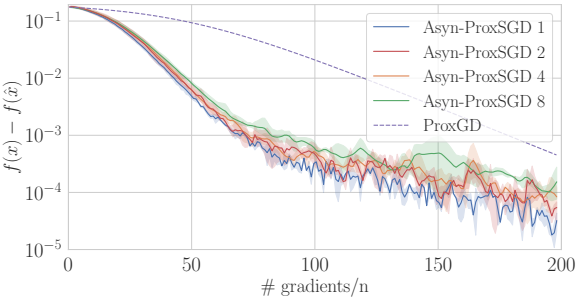
(a) `a9a`



(b) `mnist`

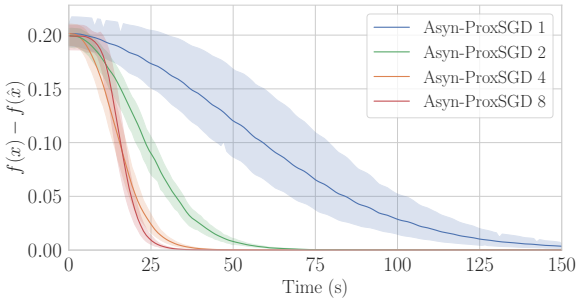


(c) `a9a`

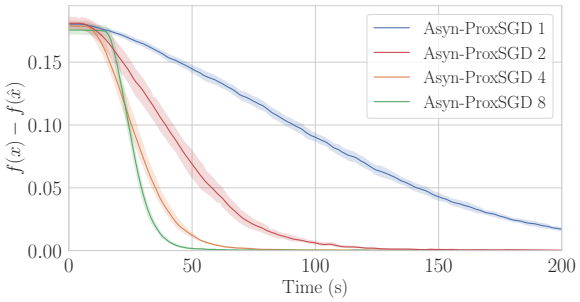


(d) `mnist`

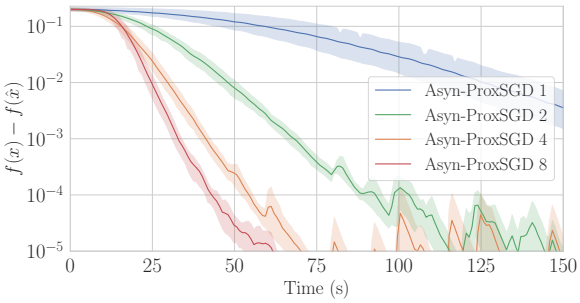
Figure 7.1: Performance of ProxGD and Async-ProxSGD on `a9a` (left) and `mnist` (right) datasets. Here the x-axis represents how many sample gradients is computed (divided by n), and the y-axis is the function suboptimality $f(\mathbf{x}) - f(\hat{\mathbf{x}})$ where $\hat{\mathbf{x}}$ is obtained by running gradient descent for many iterations with multiple restarts. Note all values on the y-axis are normalized by n.



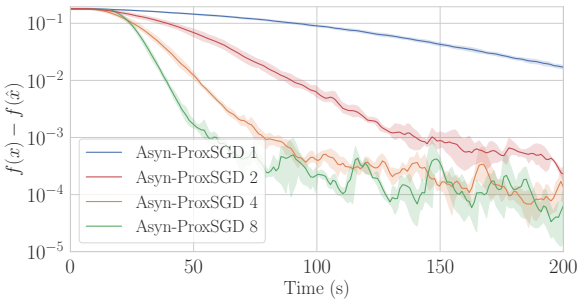
(a) `a9a`



(b) `mnist`



(c) `a9a`



(d) `mnist`

Figure 7.2: Performance of ProxGD and Async-ProxSGD on `a9a` (left) and `mnist` (right) datasets. Here the x-axis represents the actual running time, and the y-axis is the function suboptimality. Note all values on the y-axis are normalized by n.

Chapter 8

Asynchronous Block Proximal Stochastic Gradient

8.1 Background

Existing parallel algorithms fall into two categories: *data parallelism* and *model parallelism*. In data parallelism, each worker takes a subset of training samples i and calculates their loss functions f_i's and/or gradients in parallel. For example, a typical implementation of parallel SGD is to divide a minibatch with N samples into several smaller minibatches (each with N' samples), and each worker computes gradients on N' samples. This is preferred when the size of data n is large. In model parallelism, the model parameters x is partitioned into M blocks, where $x_j \in \mathbb{R}^{d_j}$ with $d_j \in \mathbb{N}_+$ and $\sum_{j=1}^{M} d_j = d$.

In the previous chapter, we have seen an example of a data parallel algorithm, Asyn-ProxSGD, for large scale nonconvex nonsmooth optimization problems. In this chapter, we propose AsyB-ProxSGD (Asynchronous Block Proximal Stochastic Gradient Descent), an extension of proximal stochastic gradient (ProxSGD) algorithm to the *model parallel* paradigm and to the *partially asynchronous protocol* (PAP) setting. In AsyB-ProxSGD, workers asynchronously communicate with the parameter servers, which collectively store model parameters in blocks. In an iteration, each worker pulls the latest yet possibly outdated model from servers, calculates partial gradients for only one block based on stochastic samples, and pushes the gradients to the corresponding server. As workers can update different blocks in parallel, AsyB-ProxSGD is different from traditional data parallel ProxSGD can handle both a large model size d and a large number n of training samples, a case frequently observed in reality.

Theoretical contributions are summarized as follows. We prove that AsyB-ProxSGD can converge to stationary points of the nonconvex and nonsmooth problem (1.1) with an ergodic convergence rate of $O(1/\sqrt{K})$, where K is the total number of times that any block in x is updated. This rate matches the convergence rate known for asynchronous SGD. The latter, however, is suitable only for smooth problems. To our best knowledge, this is the first work that provides convergence rate guarantees for ProxSGD in a model parallel mode, especially in an asynchronous setting. We also provide a linear speedup guarantee as the number of workers increases, provided that the number of workers is bounded by $O(K^{1/4})$. This result has laid down a theoretical ground for the scalability and performance of AsyB-ProxSGD in practice. Evaluation based on a real-world dataset involving both a large model and a large dataset has corroborated our theoretical findings on the convergence and speedup behavior of AsyB-ProxSGD, under a Parameter Server implementation.

The results in this chapter have appeared as technical report [7].

8.2 AsyB-ProxSGD: Asynchronous Block Proximal Stochastic Gradient

We now present *Asynchronous Block Proximal Stochastic Gradient* (AsyB-ProxSGD) algorithm. Recall that asynchronous algorithm tries to alleviate random delays in computation and communication in different iterations. When model is big, it is hard to put the whole model in a single node (a single machine or device), and we have to split it into M blocks. In this case, no single node maintains all of the parameters in memory and the nodes can update in parallel. The idea of model parallelism has been used in many applications, including deep learning [8] and factorization machine [28].

We now formally introduce how our proposed algorithm works. The main idea of our proposed algorithm is to update block x_j in parallel by different workers. In Algorithm 10, the first step is to ensure that the staleness is upper bounded by T, which is essential to ensure convergence. Here we use $\hat{\mathbf{x}}$ to emphasize that the pulled model parameters $\mathbf{x}$ may not be consistent with that stored on parameter servers. Since blocks are scattered on multiple servers, different blocks may be not consistent with updates and thus results in different delays. For example, suppose the server stores model $\mathbf{x} = (x_1, x_2)$, and we have two workers that updates x_1

and $\mathbf{x}_2$ in parallel. Our expectation is that $\mathbf{x}$ is updated by them and it becomes $\mathbf{x}' = (x_1', x_2')$. However, in partially asynchronous protocol (PAP) where workers may skip synchronization, the following case may happen. At time 1, worker 1 pushes x_1' and pulls x_2; thus, worker 1 gets (x_1', x_2). At time 2, worker 2 pushes x_2' and pulls x_1'; thus, worker 2 gets (x_1', x_2'). We can see that the next update by worker 1 is based on (x_1', x_2), which has different delays on two blocks.

Let us discuss this in more implementation details for distributed clusters. In distributed clusters, we split a large model $\mathbf{x}$ into M blocks, and one server only maintains a single block x_j to achieve model parallelism. Thus, different block may be updated at different iterations by different workers. The same phenomenon also exist in shared memory systems (i.e., a single machine with multiple CPU cores or GPUs, etc.). In these systems, the model is stored on the main memory and we can regard it as a "logical" server. In these systems, "reading" and "writing" can be done simultaneously, thus block x_j may be "pulled" while it is being updated. In summary, model parameters x may be inconsistent with any actual state on the server side.

In our algorithm, workers can update multiple blocks in parallel, and this is the spirit of model parallelism here. However, we note that on the server side, `push` request is usually more time consuming than `pull` request since it needs additional computations of the proximal operator. Therefore, we should let workers gather more stochastic gradients before pushing to the sever, and that is the reason we let each worker to compute gradients on all N samples in a minibatch. That is, a worker iteration t should compute

$$\hat{G}_{j_t}^t := \frac{1}{N} \sum_{i=1}^{N} \hat{G}_{j_t}(\hat{\mathbf{x}}^t; \xi_{i,t}),$$

where j_t is the index of block to be updated at iteration t, and $\hat{G}_{j_t}(\hat{\mathbf{x}}^t; \xi_{i,t})$ is the partial gradient w.r.t. block j_t at model $\hat{\mathbf{x}}^t$ pulled at iteration t and on sample $\xi_{i,t}$.

8.3 Convergence Analysis

To facilitate the analysis of Algorithm 10, we rewrite it in an equivalent global view (from the server's perspective), as described in Algorithm 11. In this algorithm, we define one *iteration* as the time to update any *single* block of $\mathbf{x}$ and to successfully store it at the corresponding server. We use a counter k to record how many times

Algorithm 10 AsyB-ProxSGD: Block PAP Stochastic Gradient

Server j executes:

Initialize x^0.
loop
  if `Pull Request` from a worker is received **then**
    Send x_j to the worker.
  end if
  if `Push Request` (gradient G_j) from a worker is received **then**
    $x_j \leftarrow \mathbf{prox}_{\eta h_j}(x_j - \eta G_j)$.
  end if
end loop

Worker asynchronously performs on block j:

Pull $\mathbf{x}^0$ to initialize.
for $t = 0, 1, \ldots$ **do**
  Wait until all iterations before $t - T$ are finished at all workers.
  Randomly choose N training samples indexed by $\xi_{t,1}, \ldots, \xi_{t,N}$.
  Calculate $G_j^t = \frac{1}{N} \sum_{i=1}^{N} \nabla_j F(\mathbf{x}^t; \xi_{t,i})$.
  Push G_j^t to server j.
  Pull the current model x from servers: $\mathbf{x}^{t+1} \leftarrow \mathbf{x}$.
end for

the model $\mathbf{x}$ has been updated; k increments every time a push request (model update request) is completed for a *single* block. Note that such a counter k is *not* required by workers to compute gradients and is different from the counter t in Algorithm 10—t is maintained by each worker to count how many sample gradients have been computed locally.

In particular, for every worker, it takes N stochastic sample gradients and aggregates them by averaging:

$$\hat{G}_{j_k}^k := \frac{1}{N} \sum_{i=1}^{N} \nabla_{j_k} F(\mathbf{x}^{k-\mathbf{d}_k}; \xi_{k,i}), \tag{8.1}$$

where j_k is the random index chosen at iteration k, $\mathbf{d}_k = (d_{k,1}, \ldots, d_{k,M})$ denotes the *delay vector*, i.e., the delays of different blocks in $\hat{\mathbf{x}}^k$ when computing the gradient for sample $\xi_{k,i}$ at iteration k, and $d_{k,j}$ is the delay of a specific block x_j. In addition, we denote $\hat{\mathbf{x}} := \mathbf{x}^{k-\mathbf{d}_k} := (x_1^{k-d_{k,1}}, \ldots, x_M^{k-d_{k,M}})$ as a vector of model parameters pulled from the server side. Then, the server updates $\mathbf{x}^k$ to $\mathbf{x}^{k+1}$ using proximal gradient descent.

Algorithm 11 AsyB-ProxSGD (from a Global Perspective)

1: Initialize x^1.
2: **for** $k = 1, \ldots, K$ **do**
3: Randomly select N training samples indexed by $\xi_{k,1}, \ldots, \xi_{k,N}$.
4: Randomly select a coordinate index j_k from $\{1, \ldots, M\}$.
5: Calculate the averaged gradient $\hat{G}^k_{j_k}$ according to (8.1).
6: **for** $j = 1, \ldots, M$ **do**
7: **if** $j = j_k$ **then**
8: $x_j^{k+1} \leftarrow \mathbf{prox}_{\eta_k h_j}(x_j^k - \eta_k \hat{G}_j^k)$.
9: **else**
10: $x_j^{k+1} \leftarrow x_j^k$.
11: **end if**
12: **end for**
13: **end for**

8.3.1 Assumptions and Metrics

To analyze Algorithm (11), we make some reasonable assumptions here. We assume that stochastic gradients are unbiased and with bounded variance, same with Assumption 7.2 and 7.3. We further make the following common assumptions on the delay and independence [32, 73, 114]:

Assumption 8.1 (Bounded delay)**.** *There exists an constant T such that for all k, all values in delay vector $\mathbf{d}_k$ are upper bounded by T: $0 \leq d_{k,j} \leq T$ for all j.*

Assumption 8.2 (Independence)**.** *All random variables including selected indices $\{j_k\}$ and samples $\{\xi_{k,i}\}$ for all k and i in Algorithm 11 are mutually independent.*

The assumption of bounded delay is to guarantee that gradients from workers should not be too old. Note that the maximum delay T is roughly *proportional to the number of workers* in practice. We can enforce all workers to wait for others if it runs too fast, like step 3 of workers in Algorithm 10. This setting is also called *partially synchronous parallel* [31, 84, 72] in the literature. Another assumption on independence can be met by selecting samples with *replacement*, which can be implemented using some distributed file systems like HDFS [115]. These two assumptions are common in convergence analysis for asynchronous parallel algorithms, e.g., [75, 85].

8.3.2 Theoretical Results

We present our main convergence theorem as follows:

Theorem 8.3.1. *If the step length sequence $\{\eta_k\}$ in Algorithm 10 satisfies*

$$\eta_k \leq \frac{1}{16L_{\max}}, \quad 6\eta_k L^2 T \sum_{l=1}^{T} \eta_{k+l} \leq M^2, \tag{8.2}$$

for all $k = 1, 2, \ldots, K$, we have the following ergodic convergence rate for Algorithm 11:

$$\begin{aligned} &\frac{\sum_{k=1}^{K}(\eta_k - 8L_{\max}\eta_k^2)\mathbb{E}[\|\mathcal{P}(\mathbf{x}^k)\|^2]}{\sum_{k=1}^{K}\eta_k - 8L_{\max}\eta_k^2} \\ \leq &\frac{8M(\Psi(\mathbf{x}^1) - \Psi(\mathbf{x}^*))}{\sum_{k=1}^{K}\eta_k - 8L_{\max}\eta_k^2} \\ &+ \frac{8M\sum_{k=1}^{K}\left(\frac{L\eta_k^2}{MN} + \frac{3\eta_k L^2 T\sum_{l=1}^{T}\eta_{k-l}^2}{2M^3N}\right)\sigma^2}{\sum_{k=1}^{K}\eta_k - 8L_{\max}\eta_k^2}, \end{aligned} \tag{8.3}$$

where the expectation is taken in terms of all the random variables in Algorithm 11.

Taking a closer look at Theorem 8.3.1, we can properly choose the step size η_k as a constant value and obtain the following results on convergence rate:

Corollary 8.3.1. *Let the step length be a constant, i.e.,*

$$\eta := \sqrt{\frac{(\Psi(\mathbf{x}^1) - \Psi(\mathbf{x}^*))MN}{LK\sigma^2}}. \tag{8.4}$$

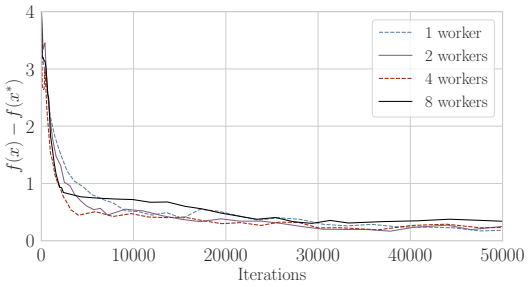
If the delay bound T satisfies

$$K \geq \frac{128(\Psi(\mathbf{x}^1) - \Psi(\mathbf{x}_*))NL}{M^3\sigma^2}(T+1)^4, \tag{8.5}$$

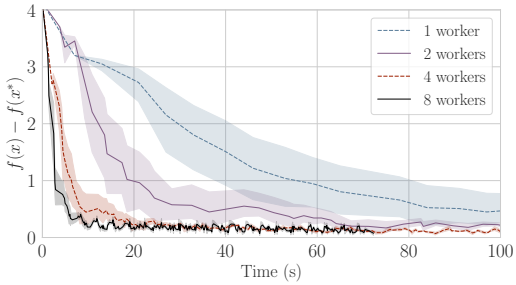
then the output of Algorithm 11 satisfies the following ergodic convergence rate as

$$\min_{k=1,\ldots,K}\mathbb{E}[\|\mathcal{P}(\mathbf{x}^k)\|^2] \leq \frac{1}{K}\sum_{k=1}^{K}\mathbb{E}[\|\mathcal{P}(\mathbf{x}^k)\|^2] \leq 32\sqrt{\frac{2(\Psi(\mathbf{x}^1) - \Psi(\mathbf{x}^*))LM\sigma^2}{KN}}. \tag{8.6}$$

Remark 1. (Linear speedup w.r.t. the staleness) When the maximum delay T is bounded by $O(K^{1/4})$, we can see that the gradient mapping $\mathbb{E}[\mathcal{P}(\mathbf{x})]$ decreases regardless of T, and thus linear speedup is achievable (if other parameters are constants). In other words, we can see that by (8.5) and (8.6), as long as T is no more than $O(K^{1/4})$, the iteration complexity (from a global perspective) to achieve ϵ-optimality is $O(1/\epsilon^2)$, which is independent from T.



(a) Iteration vs. Objective



(b) Time vs. Objective

Figure 8.1: Convergence of AsyB-ProxSGD on the sparse logistic regression problem under different numbers of workers. In this figure, the number of servers is fixed to 8.

Remark 2. (Linear speedup w.r.t. number of workers) We note that the delay bound T is roughly proportional to the number of workers, so the total iterations w.r.t. T can be an indicator of convergence w.r.t. the number of workers. As the iteration complexity is $O(1/\epsilon^2)$ to achieve ϵ-optimality, and it is independent from T, we can conclude that the total iterations will be shortened to $1/T$ of a single worker's iterations if $\Theta(T)$ workers work in parallel. This shows that our algorithm nearly achieves linear speedup.

Remark 3. (Consistency with ProxSGD) When $T = 0$, our proposed AsyB-ProxSGD reduces to the vanilla proximal stochastic gradient descent (ProxSGD) (e.g., [68]). Thus, the iteration complexity is $O(1/\epsilon^2)$ according to (8.6), attaining the same result as that in [68] *yet without assuming increased minibatch sizes.*

8.4 Experiments

We now present numerical results to confirm that our proposed algorithms can be used to solve the challenging nonconvex nonsmooth problems in machine learning.

Setup: In our experiments, we consider the sparse logistic regression problem:

$$\min_{\mathbf{x}} \frac{1}{n} \sum_{i=1}^{n} \log(1 + \exp(-b_i \cdot \mathbf{a}_i^\top \mathbf{x})) + \lambda_1 \|\mathbf{x}\| + \frac{\lambda_2}{2} \|\mathbf{x}\|^2. \tag{8.7}$$

The ℓ_1-regularized logistic regression is widely used for large scale risk minimization. We consider the `Avazu` dataset [1], which is used in a click-through rate prediction competition jointly hosted by Avazu and Kaggle in 2014. In its training dataset (`avazu-app`), there are more than 14 million samples, 1 million features, and 4050 million nonzero entries. In other words, both n and d in (8.7) are large.

We use a cluster of 16 instances on Google Cloud. Each server or worker process uses just one core. Up to 8 instances serve as server nodes, while the other 8 instances serve as worker nodes. To show the advantage of *asynchronous* parallelism, we set up four experiments adopting 1, 2, 4, and 8 worker nodes, respectively. For all experiments, the whole dataset is shuffled and all workers have a copy of this dataset. When computing a stochastic gradient, each worker takes one minibatch of random samples from its own copy. This way, each sample is used by a worker with an equal probability *empirically* to mimic the scenario of our analysis.

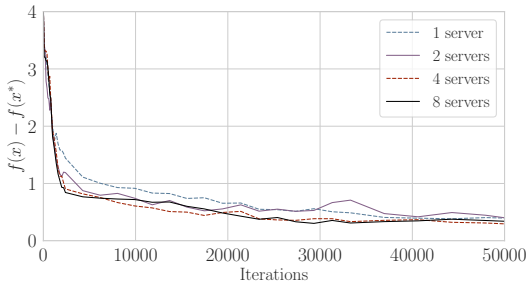
We consider the *suboptimality gap* as our performance metric, which is defined as the gap between $f(\mathbf{x})$ and $f(\mathbf{x}^*)$. Here we estimate the optimal value $\hat{\mathbf{x}}$ by performing $5\times$ as many iterations as needed for convergence. The hyper-parameters are set as follows. For all experiments, the coefficients are set as $\lambda_1 = 0.1$ and $\lambda_2 = 0.001$. We set the minibatch size to 8192. The step size is set according to $\eta_k := 0.1/\sqrt{1.0 + k}$ at iteration $k \geq 0$.

Implementation:

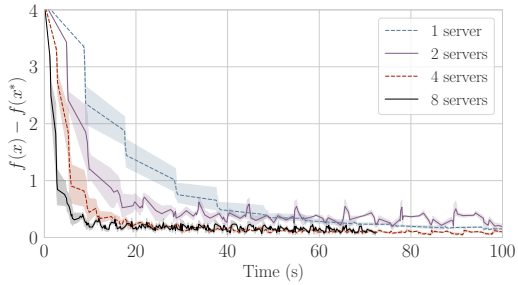
We implemented our algorithm on MXNet [9], a flexible and efficient deep learning library with support for distributed machine learning. Due to the sparse nature of the dataset, the model x is stored as a sparse `ndarray`, and in each iteration, a worker only pulls those blocks of x that are actively related to its sampled minibatch, and then calculates the gradient w.r.t. this minibatch of data, and pushes the gradients for those activated blocks only.

Results: Empirically, Assumption 8.1 (bounded delays) are observed to hold for this cluster. In our experiments, the maximum delay does not exceed 100 iterations unless some worker nodes fail. Fig. 8.1(a) and Fig. 8.1(b) show the convergence behavior of AsyB-ProxSGD algorithm in terms of objective function values. We

[1] Available at `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/`



(a) Iteration vs. Objective



(b) Time vs. Objective

Figure 8.2: Convergence of AsyB-ProxSGD on the sparse logistic regression problem under different numbers of servers. In this figure, we use 8 workers with different numbers of servers.

can clearly observe the convergence of our proposed algorithm, confirming that asynchrony with tolerable delays can still lead to convergence. In addition, the running time drops in trend when the number of workers increases.

For our proposed AsyB-ProxSGD algorithm, we are particularly interested in two kinds of speedup, namely, iteration speedup and running time speedup. If we need T_1 iterations (with T_1 sample gradients processed by servers) to achieve a certain suboptimality level using one worker, and T_p iterations to achieve the same level using p workers, then the iteration speedup is defined as $p \times T_1/T_p$ [75]. Note that iterations are counted on the server side, which is actually the number of minibatch gradients are processed by the server. On the other hand, the time speedup is simply defined as the ratio between the running time of using one worker and that of using p workers to achieve the same suboptimality level. We summarize iteration and running time speedup in Table 8.1.

We further evaluate the relationship between the number of servers and the convergence behavior. Since the model has millions of parameters to be trained, storing the whole model in a single machine can be ineffective. In fact, from Fig. 8.2 we can even see nearly linear speedup w.r.t. the number of servers. The reason here

Table 8.1: Iteration speedup and time speedup of AsyB-ProxSGD at the optimality level 10^{-1}.

Workers	1	2	4	8
Iteration Speedup	1.000	2.127	3.689	6.748
Time Speedup	1.000	1.973	4.103	8.937

is that, more servers can significantly decrease the length of request queue at the server side. When we have only one server, the blue dashed curve in Fig. 8.2(b) looks like a tilt staircase, and further investigation shows that some `push` requests take too long time to be processed. Therefore, we have to set more than one servers to observe parallel speedup in Fig. 8.1 so that servers are not the bottleneck.

8.5 Proof of Theorem 8.3.1

To simplify notations, we use j instead of j_k in this section. Since we update one block only in each iteration, we define an auxiliary function as follows:

$$P_j(\mathbf{x}, g, \eta) := \frac{1}{\eta}(x_j - \mathbf{prox}_{\eta h_j}(x_j - \eta g_j)),$$

where the variables x_j and g_j take the j-th coordinate. Our proof of Theorem 8.3.1 relies on the following milestone lemmas.

8.5.1 Milestone Lemmas

Lemma 8.5.1 (Descent Lemma)**.**

$$\mathbb{E}_j[\Psi(\mathbf{x}^{k+1})|\mathcal{F}_k] \leq \mathbb{E}_j[\Psi(\mathbf{x}^k)|\mathcal{F}_k] - \frac{\eta_k - 4L_{\max}\eta_k^2}{2M}\|P(\mathbf{x}^k, \hat{g}^k, \eta_k)\|^2 + \frac{\eta_k}{2M}\|g^k - \hat{g}^k\|^2 + \frac{L\eta_k^2}{MN}\sigma^2. \tag{8.8}$$

Lemma 8.5.2. *Suppose we have a sequence $\{\mathbf{x}^k\}$ by Algorithm 11. Then, we have*

$$\mathbb{E}[\|\mathbf{x}^k - \mathbf{x}^{k-\tau}\|^2] \leq \frac{2T\sum_{l=1}^T \eta_{k-l}^2}{MN}\sigma^2 + 2\left\|\sum_{l\in K(\boldsymbol{\tau}(k))} \eta_{k-l} P_{j_{k-l}}(\mathbf{x}^{k-l}, \hat{g}^{k-l}, \eta_{k-l})\right\|^2 \tag{8.9}$$

Lemma 8.5.3. *Suppose we have a sequence $\{\mathbf{x}^k\}$ by Algorithm 11. Then, we have*

$$\mathbb{E}[\|g_j^k - \hat{g}_j^k\|^2] \leq \frac{2L^2T\sum_{l=1}^T \eta_{k-l}^2}{M^2N}\sigma^2 + \frac{2L^2T}{M^2}\sum_{l=1}^T \eta_{k-l}^2\|P(\mathbf{x}^{k-l}, \hat{g}^{k-l}, \eta_{k-l})\|^2 \tag{8.10}$$

Proof of Theorem 8.3.1. We have

$$\|P_j(\mathbf{x}^k, \tilde{g}_k, \eta_k)\|^2 \geq \frac{1}{2}\|P_j(\mathbf{x}^k, g_k, \eta_k)\|^2 - \|g_k - \tilde{g}_k\|^2.$$

When we have $\eta_k \leq \frac{1}{8L_{\max}}$, we can apply the above equation following Lemma 8.5.1:

$$\begin{aligned}
&\mathbb{E}_j[\Psi(\mathbf{x}^{k+1})|\mathcal{F}_k] \\
&\leq \mathbb{E}_j[\Psi(\mathbf{x}^k)|\mathcal{F}_k] - \frac{\eta_k - 4L_{\max}\eta_k^2}{2M}\|P(\mathbf{x}^k, \hat{g}^k, \eta_k)\|^2 + \frac{\eta_k}{2M}\|g^k - \hat{g}^k\|^2 + \frac{L\eta_k^2}{MN}\sigma^2 \\
&\leq \mathbb{E}_j[\Psi(\mathbf{x}^k)|\mathcal{F}_k] - \frac{\eta_k - 8L_{\max}\eta_k^2}{4M}\|P(\mathbf{x}^k, \hat{g}^k, \eta_k)\|^2 - \frac{\eta_k}{4M}\|P(\mathbf{x}^k, \hat{g}^k, \eta_k)\|^2 \\
&\quad + \frac{\eta_k}{2M}\|g^k - \hat{g}^k\|^2 + \frac{L\eta_k^2}{MN}\sigma^2 \\
&\leq \mathbb{E}_j[\Psi(\mathbf{x}^k)|\mathcal{F}_k] - \frac{\eta_k - 8L_{\max}\eta_k^2}{8M}\|P(\mathbf{x}^k, g^k, \eta_k)\|^2 + \frac{3\eta_k}{4M}\|g^k - \hat{g}^k\|^2 \\
&\quad - \frac{\eta_k}{4M}\|P(\mathbf{x}^k, \hat{g}^k, \eta_k)\|^2 + \frac{L\eta_k^2}{MN}\sigma^2
\end{aligned}$$

By Lemma 8.5.3, we have

$$\begin{aligned}
&\mathbb{E}_j[\Psi(\mathbf{x}^{k+1})|\mathcal{F}_k] \\
&\leq \mathbb{E}_j[\Psi(\mathbf{x}^k)|\mathcal{F}_k] - \frac{\eta_k - 8L_{\max}\eta_k^2}{8M}\|P(\mathbf{x}^k, g^k, \eta_k)\|^2 - \frac{\eta_k}{4M}\|P(\mathbf{x}^k, \hat{g}^k, \eta_k)\|^2 + \frac{L\eta_k^2}{MN}\sigma^2 \\
&\quad + \frac{3\eta_k}{4M}\left(\frac{2L^2T\sum_{l=1}^T \eta_{k-l}^2}{M^2N}\sigma^2 + \frac{2L^2T}{M^2}\sum_{l=1}^T \eta_{k-l}^2\|P(\mathbf{x}^{k-l}, \hat{g}^{k-l}, \eta_{k-l})\|^2\right) \\
&\leq \mathbb{E}_j[\Psi(\mathbf{x}^k)|\mathcal{F}_k] - \frac{\eta_k - 8L_{\max}\eta_k^2}{8M}\|P(\mathbf{x}^k, g^k, \eta_k)\|^2 + \left(\frac{L\eta_k^2}{MN} + \frac{3\eta_k L^2T\sum_{l=1}^T \eta_{k-l}^2}{2M^3N}\right)\sigma^2 \\
&\quad + \frac{3\eta_k L^2T}{2M^3}\sum_{l=1}^T \eta_{k-l}^2\|P_{j_{k-l}}(\mathbf{x}^{k-l}, \hat{g}^{k-l}, \eta_{k-l})\|^2 - \frac{\eta_k}{4M}\|P(\mathbf{x}^k, \hat{g}^k, \eta_k)\|^2
\end{aligned}$$

By taking telescope sum, we have

$$\begin{aligned}
&\sum_{k=1}^K \frac{\eta_k - 8L_{\max}\eta_k^2}{8M}\|P(\mathbf{x}^k, g^k, \eta_k)\|^2 \\
&\leq \Psi(\mathbf{x}^1) - \Psi(\mathbf{x}^*) + \sum_{k=1}^K\left(\frac{L\eta_k^2}{MN} + \frac{3\eta_k L^2T\sum_{l=1}^T \eta_{k-l}^2}{2M^3N}\right)\sigma^2 \\
&\quad - \sum_{k=1}^K(\frac{\eta_k}{4M} - \frac{3\eta_k^2L^2}{2M^3}\sum_{l=1}^{l_k}\eta_{k+l}^2)\|P(\mathbf{x}^k, \hat{g}^k, \eta_k)\|^2 \\
&\leq \Psi(\mathbf{x}^1) - \Psi(\mathbf{x}^*) + \sum_{k=1}^K\left(\frac{L\eta_k^2}{MN} + \frac{3\eta_k L^2T\sum_{l=1}^T \eta_{k-l}^2}{2M^3N}\right)\sigma^2,
\end{aligned}$$

where the last inequality follows from the assumption that $6\eta_k^2L^2\sum_{l=1}^T \eta_{k+l} \leq M^2$, and now we prove Theorem 8.3.1.

□

8.6 Proof of Corollary 8.3.1

Proof. Since the learning rate $\eta_k := \eta$ is a constant, we apply it to Theorem 8.3.1 and we have:

$$\frac{1}{K}\sum_{k=1}^{K}\mathbb{E}[\|P(\mathbf{x}^k, g^k, \eta_k)\|^2] \leq \frac{16M(\Psi(\mathbf{x}^1) - \Psi(\mathbf{x}^*))}{K\eta} + \frac{16M}{\eta}\left(\frac{L\eta^2}{MN} + \frac{3L^2T^2\eta^3}{2M^3N}\right)\sigma^2. \tag{8.11}$$

Following conditions in Corollary 8.3.1, we have

$$\eta \leq \frac{M^2}{16L(T+1)^2},$$

and thus we have

$$\begin{aligned}\frac{3LT^2\eta}{2M^2} &\leq \frac{3M^2T^2}{2M^2\cdot 16(T+1)^2} \leq 1,\\ \frac{3L^2T^2\eta^3}{2M^2} &\leq L\eta^2.\end{aligned}$$

Then, we can estimate (8.11) from the above inequality as follows:

$$\begin{aligned}\frac{1}{K}\sum_{k=1}^{K}\mathbb{E}[\|P(\mathbf{x}^k, g^k, \eta_k)\|^2] &\leq \frac{16M(\Psi(\mathbf{x}^1) - \Psi(\mathbf{x}^*))}{K\eta} + \frac{16M}{\eta}\left(\frac{L\eta^2}{MN} + \frac{3L^2T^2\eta^3}{2M^3N}\right)\sigma^2\\ &\leq \frac{16M(\Psi(\mathbf{x}^1) - \Psi(\mathbf{x}^*))}{K\eta} + \frac{32L\eta}{N}\sigma^2\\ &= 32\sqrt{\frac{2(\Psi(\mathbf{x}^1) - \Psi(\mathbf{x}^*))LM(T+1)\sigma^2}{KN}},\end{aligned}$$

which proves the corollary. □

8.7 Proof of Milestone Lemmas

Proof of Lemma 8.5.1. Recall Corollary A.1.4:

$$\begin{aligned}\Psi(\mathbf{x}^{k+1}) \leq \Psi(\bar{\mathbf{x}}^{k+1}) &+ \langle \nabla_j f(\mathbf{x}^k) - \hat{G}_j^k, x_j^{k+1} - \bar{x}_j^{k+1}\rangle\\ &+ \left(\frac{L_j}{2} - \frac{1}{2\eta}\right)\|y_j - x_j\|^2 + \left(\frac{L_j}{2} + \frac{1}{2\eta}\right)\|z_j - x_j\|^2 - \frac{1}{2\eta}\|y_j - x_j\|^2.\end{aligned} \tag{8.12}$$

Now we turn to bound $\Psi(\bar{x}_j^{k+1})$ as follows:

$$
\begin{aligned}
& f(\bar{\mathbf{x}}^{k+1}) \\
\leq\ & f(\mathbf{x}^k) + \langle \nabla_j f(\mathbf{x}^k), \bar{x}_j^{k+1} - x_j^k \rangle + \frac{L_j}{2}\|\bar{x}_j^{k+1} - x_j^k\|^2 \\
=\ & f(\mathbf{x}^k) + \langle g_j^k, \bar{x}_j^{k+1} - x_j^k \rangle + \frac{\eta_k^2 L_j}{2}\|P_j(\mathbf{x}^k, \hat{g}^k, \eta_k)\|^2 \\
=\ & f(\mathbf{x}^k) + \langle \hat{g}_j^k, \bar{x}_j^{k+1} - x_j^k \rangle + \langle g_j^k - \hat{g}_j^k, \bar{x}_j^{k+1} - x_j^k \rangle + \frac{\eta_k^2 L_j}{2}\|P_j(\mathbf{x}^k, \hat{g}^k, \eta_k)\|^2 \\
=\ & f(\mathbf{x}^k) - \eta_k \langle \hat{g}_j^k, P_j(\mathbf{x}^k, \hat{g}_j^k, \eta_k) \rangle + \langle g_j^k - \hat{g}_j^k, \bar{x}_j^{k+1} - x_j^k \rangle + \frac{\eta_k^2 L_j}{2}\|P_j(\mathbf{x}^k, \hat{g}^k, \eta_k)\|^2 \\
\leq\ & f(\mathbf{x}^k) - [\eta_k \|P_j(\mathbf{x}^k, \hat{g}^k, \eta_k)\|^2 + h_j(\bar{x}_j^{k+1}) - h_j(x_j^k)] \\
& + \langle g_j^k - \hat{g}_j^k, \bar{x}_j^{k+1} - x_j^k \rangle + \frac{\eta_k^2 L_j}{2}\|P_j(\mathbf{x}^k, \hat{g}^k, \eta_k)\|^2,
\end{aligned}
$$

where the last inequality follows from Corollary A.1.1. By rearranging terms on both sides, we have

$$
\Psi(\bar{\mathbf{x}}^{k+1}) \leq \Psi(\mathbf{x}^k) - (\eta_k - \frac{\eta_k^2 L_j}{2})\|P_j(\mathbf{x}^k, \hat{g}^k, \eta_k)\|^2 + \langle g_j^k - \hat{g}_j^k, \bar{x}_j^{k+1} - x_j^k \rangle \tag{8.13}
$$

Taking the summation of (8.12) and (8.13), we have

$$
\begin{aligned}
& \Psi(\mathbf{x}^{k+1}) \\
\leq\ & \Psi(\mathbf{x}^k) + \langle \nabla_j f(\mathbf{x}^k) - \hat{G}_j^k, x_j^{k+1} - \bar{x}_j^{k+1} \rangle \\
& + \left(\frac{L_j}{2} - \frac{1}{2\eta}\right)\|y_j - x_j\|^2 + \left(\frac{L_j}{2} + \frac{1}{2\eta_k}\right)\|z_j - x_j\|^2 - \frac{1}{2\eta_k}\|y_j - x_j\|^2 \\
& - \left(\eta_k - \frac{\eta_k^2 L_j}{2}\right)\|P_j(\mathbf{x}^k, \hat{g}_k, \eta_k)\|^2 + \langle g_j^k - \hat{g}_j^k, \bar{x}_j^{k+1} - x_j^k \rangle \\
=\ & \Psi(\mathbf{x}^k) + \langle \nabla_j f(\mathbf{x}^k) - \hat{g}_j^k, x_j^{k+1} - \bar{x}_j^{k+1} \rangle + \langle \hat{g}_j^k - \hat{G}_j^k, x_j^{k+1} - \bar{x}_j^{k+1} \rangle \\
& + \left(\frac{L_j \eta_k^2}{2} - \frac{\eta_k}{2}\right)\|P_j(\mathbf{x}^k, \hat{G}^k, \eta_k)\|^2 + \left(\frac{L_j \eta_k^2}{2} + \frac{\eta_k}{2}\right)\|P_j(\mathbf{x}^k, \hat{g}^k, \eta_k)\|^2 \\
& - \frac{1}{2\eta_k}\|x_j^{k+1} - \bar{x}_j^{k+1}\|^2 - (\eta_k - \frac{\eta_k^2 L_j}{2})\|P_j(\mathbf{x}^k, \hat{g}^k, \eta_k)\|^2 \\
=\ & \Psi(\mathbf{x}^k) + \langle x_j^{k+1} - x_j^k, g_j^k - \hat{g}_j^k \rangle + \langle x_j^{k+1} - \bar{x}_j^{k+1}, \delta_j^k \rangle + \frac{L_j \eta_k^2 - \eta_k}{2}\|P_j(\mathbf{x}^k, \hat{G}^k, \eta_k)\|^2 \\
& + \frac{2L_j \eta_k^2 - \eta_k}{2}\|P_j(\mathbf{x}^k, \hat{g}^k, \eta_k)\|^2 - \frac{1}{2\eta_k}\|x_j^{k+1} - \bar{x}_j^{k+1}\|^2.
\end{aligned}
$$

By taking the expectation on condition of filtration $\mathcal{F}_k$ and j, we have the following equation according to Assumption 7.2:

$$
\begin{aligned}
& \mathbb{E}_j[\Psi(\mathbf{x}^{k+1})|\mathcal{F}_k] \\
\leq\ & \mathbb{E}_j[\Psi(\mathbf{x}^k)|\mathcal{F}_k] + \frac{1}{M}\mathbb{E}[\langle \mathbf{x}^{k+1} - \mathbf{x}^k, g^k - \hat{g}_k \rangle|\mathcal{F}_k] + \frac{L_{\max}\eta_k^2 - \eta_k}{2M}\mathbb{E}[\|P(\mathbf{x}^k, \hat{G}^k, \eta_k)\|^2|\mathcal{F}_k] \\
& + \frac{2L_{\max}\eta_k^2 - \eta_k}{2M}\|P(\mathbf{x}^k, \hat{g}^k, \eta_k)\|^2 - \frac{1}{2M\eta_k}\|\mathbf{x}^{k+1} - \bar{\mathbf{x}}^{k+1}\|^2.
\end{aligned} \tag{8.14}
$$

Therefore, we have

$$
\begin{aligned}
&\mathbb{E}_j[\Psi(\mathbf{x}^{k+1})|\mathcal{F}_k] \\
&\leq \mathbb{E}_j[\Psi(\mathbf{x}^k)|\mathcal{F}_k] + \frac{1}{M}\mathbb{E}[\langle \mathbf{x}^{k+1}-\mathbf{x}^k, g^k-\hat{g}^k\rangle|\mathcal{F}_k] + \frac{L_{\max}\eta_k^2-\eta_k}{2M}\mathbb{E}[\|P(\mathbf{x}^k,\hat{G}^k,\eta_k)\|^2|\mathcal{F}_k] \\
&\quad + \frac{2L_{\max}\eta_k^2-\eta_k}{2M}\|P(\mathbf{x}^k,\hat{g}^k,\eta_k)\|^2 - \frac{1}{2M\eta_k}\|\mathbf{x}^{k+1}-\bar{\mathbf{x}}^{k+1}\|^2 \\
&\leq \mathbb{E}_j[\Psi(\mathbf{x}^k)|\mathcal{F}_k] + \frac{\eta_k}{2M}\|g^k-\hat{g}^k\|^2 + \frac{L_{\max}\eta_k^2}{2M}\mathbb{E}[\|P(\mathbf{x}^k,\hat{G}^k,\eta_k)\|^2|\mathcal{F}_k] + \frac{2L_{\max}\eta_k^2-\eta_k}{2M}\|P(\mathbf{x}^k,\hat{g}^k,\eta_k)\|^2 \\
&\leq \mathbb{E}_j[\Psi(\mathbf{x}^k)|\mathcal{F}_k] \\
&\quad - \frac{\eta_k-4L_{\max}\eta_k^2}{2M}\|P(\mathbf{x}^k,\hat{g}^k,\eta_k)\|^2 + \frac{\eta_k}{2M}\|g^k-\hat{g}^k\|^2 + \frac{L\eta_k^2}{MN}\sigma^2.
\end{aligned}
$$

□

Proof of Lemma 8.5.2.

$$
\begin{aligned}
\|\mathbf{x}^k-\mathbf{x}^{k-\tau}\|^2 &= \left\|\sum_{l\in K(\boldsymbol{\tau}(k))}\mathbf{x}^{k-l+1}-\mathbf{x}^{k-l}\right\|^2 \\
&= \left\|\sum_{l\in K(\boldsymbol{\tau}(k))}\eta_{k-l}P_{j_{k-l}}(\mathbf{x}^{k-l},\hat{G}^{k-l},\eta_{k-l})\right\|^2 \\
&\leq 2\left\|\sum_{l\in K(\boldsymbol{\tau}(k))}\eta_{k-l}(P_{j_{k-l}}(\mathbf{x}^{k-l},\hat{G}^{k-l},\eta_{k-l})-P_{j_{k-l}}(\mathbf{x}^{k-l},\hat{g}^{k-l},\eta_{k-l}))\right\|^2 \\
&\quad + 2\left\|\sum_{l\in K(\boldsymbol{\tau}(k))}\eta_{k-l}P_{j_{k-l}}(\mathbf{x}^{k-l},\hat{g}^{k-l},\eta_{k-l})\right\|^2 \\
&\leq 2T\sum_{l\in K(\boldsymbol{\tau}(k))}\eta_{k-l}^2\left\|P_{j_{k-l}}(\mathbf{x}^{k-l},\hat{G}^{k-l},\eta_{k-l})-P_{j_{k-l}}(\mathbf{x}^{k-l},\hat{g}^{k-l},\eta_{k-l})\right\|^2 \\
&\quad + 2\left\|\sum_{l\in K(\boldsymbol{\tau}(k))}\eta_{k-l}P_{j_{k-l}}(\mathbf{x}^{k-l},\hat{g}^{k-l},\eta_{k-l})\right\|^2 \\
&\leq 2T\sum_{l=1}^{T}\eta_{k-l}^2\|\hat{G}^{k-l}-\hat{g}^{k-l}\|^2 + 2\left\|\sum_{l\in K(\boldsymbol{\tau}(k))}\eta_{k-l}P_{j_{k-l}}(\mathbf{x}^{k-l},\hat{g}^{k-l},\eta_{k-l})\right\|^2 \\
&\leq \frac{2T\sum_{l=1}^{T}\eta_{k-l}^2}{MN}\sigma^2 + 2\left\|\sum_{l\in K(\boldsymbol{\tau}(k))}\eta_{k-l}P_{j_{k-l}}(\mathbf{x}^{k-l},\hat{g}^{k-l},\eta_{k-l})\right\|^2
\end{aligned}
$$

□

Proof of Lemma 8.5.3.

$$
\begin{aligned}
\mathbb{E}[\|g_j^k - \hat{g}_j^k\|^2] &= \left\| \frac{1}{N} \sum_{i=1}^{N} g_j^k - \hat{g}_j^{k-\tau(k,i)} \right\|^2 \\
&\leq \frac{1}{N} \sum_{i=1}^{N} \|g_j^k - \hat{g}_j^{k-\tau(k,i)}\|^2 \\
&\leq \frac{1}{MN} \sum_{i=1}^{N} \|g^k - \hat{g}^{k-\tau(k,i)}\|^2 \\
&\leq \frac{L^2}{MN} \sum_{i=1}^{N} \|\mathbf{x}^k - \hat{\mathbf{x}}^{k-\tau(k,i)}\|^2 \\
&\leq \frac{L^2}{MN} \sum_{i=1}^{N} \left(\frac{2T \sum_{l=1}^{T} \eta_{k-l}^2}{MN} \sigma^2 + 2 \left\| \sum_{l=1}^{T} \eta_{k-l} P_{j_{k-l}}(\mathbf{x}^{k-l}, \hat{g}^{k-l}, \eta_{k-l}) \right\|^2 \right) \\
&\leq \frac{2L^2 T \sum_{l=1}^{T} \eta_{k-l}^2}{M^2 N} \sigma^2 + \frac{2L^2 T}{M^2} \sum_{l=1}^{T} \eta_{k-l}^2 \|P(\mathbf{x}^{k-l}, \hat{g}^{k-l}, \eta_{k-l})\|^2
\end{aligned}
$$

□

Chapter 9

Conclusion

This chapter summarizes the main contributions of the previous chapters and discuss relevant work for each.

In Chapter 3, we propose a robust class of matrix completion algorithms, called IS-p, to approximate the rank minimization problem with reweighted Schatten-p norm minimization, and prove that the algorithm can converge for any p between 1 and 2. We further enhance the latency prediction with the help of partially collected historical observations forming a tensor, and extend our IS-p algorithm to the case of approximate tensor completion. Extensive evaluations based on the Seattle data show that our proposed algorithms outperform state-of-the-art techniques, including network embedding (e.g., high-dimensional Vivaldi with/without heights) and matrix factorization (e.g., DMFSGD) by a substantial margin, although they do not show much improvement on traditional PlanetLab data. This reveals the fact that our algorithms can better estimate latencies in personal device networks, for which traditional schemes are insufficient due to triangle inequality violation, asymmetric latencies and time-varying characteristics. The prediction accuracy is further significantly improved by exploiting the inherent autocorrelation property in the data sampled over multiple periods, through the proposed approximate tensor completion scheme.

In Chapter 4 and Chapter 5, we propose novel matrix factorization approaches for skewed data, namely, Quantile Matrix Factorization (QMF) and Expectile Matrix Factorization (EMF). Compared with existing popular matrix factorization approaches, which aim at minimizing the mean squared error and actually estimate the conditional means of the values, we propose the QMF and EMF which novelly combine the regression techniques and matrix factorization. In particular, we

propose an efficient algorithm based on Iterative Reweighted Least Squares (IRLS) to solve QMF in Chapter 4. In this chapter, extensive evaluations based on a real-world dataset of web service QoS measurements show that QMF significantly outperforms several state-of-the-art QoS prediction and recommendation algorithms based on matrix factorization or collaborative filtering, especially in terms of excluding services with the highest response times and selecting the best services. Furthermore, the prediction and ranking performance of QMF are particularly robust to the skewed QoS data.

In Chapter 5, we propose EMF which introduces the "asymmetric least squares" loss function of expectile regression analysis originated in statistics and econometrics into matrix factorization for robust matrix estimation. Existing matrix factorization techniques aim at minimizing the mean squared error and essentially estimate the conditional means of matrix entries. In contrast, the proposed EMF can yield the ωth conditional expectile estimates of matrix entries for any $\omega \in (0, 1)$, accommodating the conventional matrix factorization as a special case of $\omega = 0.5$. We propose an efficient alternating minimization algorithm to solve EMF and theoretically prove its convergence to the global optimality in the noiseless case. Through evaluation based on both synthetic data and a dataset containing real-world web service response times, we show that EMF achieves better recovery than conventional matrix factorization when the data is skewed or contaminated by skewed noise. By using a flexible ω, EMF is not only more robust to outliers but can also be tuned to obtain a more comprehensive understanding of data distribution in a matrix, depending on application requirements.

We study distributed nonconvex nonsmooth optimization problems in Chapter 6 and Chapter 7. In Chapter 6, we propose a block-wise, asynchronous and distributed ADMM algorithm to solve general nonconvex and nonsmooth optimization problems in machine learning. Under the bounded delay assumption, we have shown that our proposed algorithm can converge to stationary points satisfying KKT conditions. The block-wise updating nature of our algorithm makes it feasible to be implemented on Parameter Server, take advantage of the ability to update different blocks of all model parameters in parallel on distributed servers. Experimental results based on a real-world dataset have demonstrated the convergence and near-linear speedup of the proposed ADMM algorithm, for training large-scale sparse logistic regression models in Amazon EC2 clusters.

In Chapter 7 and 8, we study asynchronous parallel implementations of stochastic proximal gradient methods for solving nonconvex optimization problems, with convex yet possibly nonsmooth regularization. Compared to asynchronous parallel stochastic gradient descent (Asyn-SGD), which is targeting smooth optimization, the understanding of the convergence and speedup behavior of stochastic algorithms for the nonsmooth regularized optimization problems is quite limited, especially when the objective function is nonconvex. To fill this gap, we propose an asynchronous proximal stochastic gradient descent algorithms with convergence rates provided for nonconvex problems.

In particular, we propose two types of algorithms for different parallelisms. Existing parallel algorithms fall into two categories, namely, *data parallelism* and *model parallelism*, which focuses on dealing with large scales in data volume and model dimension, respectively. In Chapter 7, we propose Asyn-ProxSGD, in which workers calculate an averaged gradient from a minibatch in parallel, a *data parallel* algorithm. In Chapter 8, we propose another algorithm AsyB-ProxSGD that allow workers update different blocks of the model in parallel, which is a *model parallel* algorithm. Our theoretical analysis suggests that the same order of convergence rate can be achieved for asynchronous ProxSGD for nonsmooth problems as for the asynchronous SGD, under constant minibatch sizes, without making additional assumptions on variance reduction. And a linear speedup is proven to be achievable for both asynchronous ProxSGD when the number of workers is bounded by $O(K^{1/4})$.

Bibliography

[1] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004.

[2] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends® in Machine Learning*, vol. 3, no. 1, pp. 1–122, 2011.

[3] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*. Prentice Hall, 1989.

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

[5] R. Zhu, D. Niu, and Z. Li, "A block-wise, asynchronous and distributed admm algorithm for general form consensus optimization," 2018, arXiv:1802.08882.

[6] ——, "Asynchronous stochastic proximal methods for nonconvex nonsmooth optimization," 2018, arXiv:1802.08880.

[7] R. Zhu and D. Niu, "A model parallel proximal stochastic gradient algorithm for partially asynchronous systems," 2018, submitted to AISTATS 2019.

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

[9] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang, "Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems," *arXiv preprint arXiv:1512.01274*, 2015.

[10] S. Zhang, A. E. Choromanska, and Y. LeCun, "Deep learning with elastic averaging sgd," in *Advances in Neural Information Processing Systems*, 2015, pp. 685–693.

[11] R. Tibshirani, M. Saunders, S. Rosset, J. Zhu, and K. Knight, "Sparsity and smoothness via the fused lasso," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 67, no. 1, pp. 91–108, 2005.

[12] J. Liu, J. Chen, and J. Ye, "Large-scale sparse logistic regression," in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2009, pp. 547–556.

[13] H. Xu, C. Caramanis, and S. Sanghavi, "Robust pca via outlier pursuit," in *Advances in Neural Information Processing Systems*, 2010, pp. 2496–2504.

[14] R. Sun and Z.-Q. Luo, "Guaranteed matrix completion via nonconvex factorization," in *Foundations of Computer Science (FOCS), 2015 IEEE 56th Annual Symposium on*. IEEE, 2015, pp. 270–289.

[15] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[16] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su, "Scaling distributed machine learning with the parameter server," in *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, 2014, pp. 583–598.

[17] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, and J. M. Hellerstein, "Distributed graphlab: a framework for machine learning and data mining in the cloud," *Proceedings of the VLDB Endowment*, vol. 5, no. 8, pp. 716–727, 2012.

[18] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. USENIX Association, 2016, pp. 265–283.

[19] Y. Nesterov, *Introductory lectures on convex optimization: A basic course*. Springer Science & Business Media, 2013, vol. 87.

[20] D. P. Bertsekas, *Nonlinear Programming*. Athena Scientific, 2016.

[21] N. Parikh, S. Boyd *et al.*, "Proximal algorithms," *Foundations and Trends® in Optimization*, vol. 1, no. 3, pp. 127–239, 2014.

[22] H. Attouch, J. Bolte, and B. F. Svaiter, "Convergence of descent methods for semi-algebraic and tame problems: proximal algorithms, forward–backward splitting, and regularized gauss–seidel methods," *Mathematical Programming*, vol. 137, no. 1-2, pp. 91–129, 2013.

[23] K. G. Murty and S. N. Kabadi, "Some np-complete problems in quadratic and nonlinear programming," *Mathematical programming*, vol. 39, no. 2, pp. 117–129, 1987.

[24] S. Ghadimi and G. Lan, "Stochastic first-and zeroth-order methods for nonconvex stochastic programming," *SIAM Journal on Optimization*, vol. 23, no. 4, pp. 2341–2368, 2013.

[25] S. J. Reddi, S. Sra, B. Póczos, and A. J. Smola, "Proximal stochastic methods for nonsmooth nonconvex finite-sum optimization," in *Advances in Neural Information Processing Systems*, 2016, pp. 1145–1153.

[26] D. Garber, E. Hazan, C. Jin, S. M. Kakade, C. Musco, P. Netrapalli, and A. Sidford, "Faster eigenvector computation via shift-and-invert preconditioning." in *ICML*, 2016, pp. 2626–2634.

[27] D. P. Bertsekas, *Convex optimization theory*. Athena Scientific Belmont, 2009.

[28] M. Li, Z. Liu, A. J. Smola, and Y.-X. Wang, "Difacto: Distributed factorization machines," in *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*. ACM, 2016, pp. 377–386.

[29] A. Ahmed, M. Aly, J. Gonzalez, S. Narayanamurthy, and A. J. Smola, "Scalable inference in latent variable models," in *Proceedings of the fifth ACM international conference on Web search and data mining*. ACM, 2012, pp. 123–132.

[30] A. Agarwal and J. C. Duchi, "Distributed delayed stochastic optimization," in *Advances in Neural Information Processing Systems*, 2011, pp. 873–881.

[31] Q. Ho, J. Cipar, H. Cui, S. Lee, J. K. Kim, P. B. Gibbons, G. A. Gibson, G. Ganger, and E. P. Xing, "More effective distributed ml via a stale synchronous parallel parameter server," in *Advances in neural information processing systems*, 2013, pp. 1223–1231.

[32] F. Niu, B. Recht, C. Re, and S. Wright, "Hogwild: A lock-free approach to parallelizing stochastic gradient descent," in *Advances in Neural Information Processing Systems*, 2011, pp. 693–701.

[33] X. Pan, M. Lam, S. Tu, D. Papailiopoulos, C. Zhang, M. I. Jordan, K. Ramchandran, and C. Ré, "Cyclades: Conflict-free asynchronous machine learning," in *Advances in Neural Information Processing Systems*, 2016, pp. 2568–2576.

[34] C. J. Hillar and L.-H. Lim, "Most Tensor Problems Are NP-Hard," *Journal of the ACM (JACM)*, vol. 60, no. 6, p. 45, 2013.

[35] M. Fazel, "Matrix rank minimization with applications," Ph.D. dissertation, PhD thesis, Stanford University, 2002.

[36] B. Recht, M. Fazel, and P. A. Parrilo, "Guaranteed Minimum-Rank Solutions of Linear Matrix Equations via Nuclear Norm Minimization," *SIAM Review*, vol. 52, no. 3, pp. 471–501, 2010.

[37] J.-F. Cai, E. J. Candès, and Z. Shen, "A singular value thresholding algorithm for matrix completion," *SIAM Journal on Optimization*, vol. 20, no. 4, pp. 1956–1982, 2010.

[38] J. Liu, P. Musialski, P. Wonka, and J. Ye, "Tensor completion for estimating missing values in visual data," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 1, pp. 208–220, 2013.

[39] R. Zhu, B. Liu, D. Niu, Z. Li, and H. V. Zhao, "Network latency estimation for personal devices: A matrix completion approach," *IEEE/ACM Transactions on Networking*, vol. 25, no. 2, pp. 724–737, 2017.

[40] R. Zhu, D. Niu, and Z. Li, "Robust web service recommendation via quantile matrix factorization," in *Proc. IEEE INFOCOM*, Atlanta, GA, USA, May 2017.

[41] R. Zhu, D. Niu, L. Kong, and Z. Li, "Expectile matrix factorization for skewed data analysis," in *AAAI Conference on Artificial Intelligence*, 2017.

[42] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, no. 8, pp. 30–37, 2009.

[43] Y. Liao, W. Du, P. Geurts, and G. Leduc, "DMFSGD: A Decentralized Matrix Factorization Algorithm for Network Distance Prediction," *IEEE/ACM Trans. Netw. (TON)*, vol. 21, no. 5, pp. 1511–1524, 2013.

[44] P. Chen and D. Suter, "Recovering the missing components in a large noisy low-rank matrix: Application to SFM," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 26, no. 8, pp. 1051–1063, 2004.

[45] Z. Liu and L. Vandenberghe, "Interior-point method for nuclear norm approximation with application to system identification," *SIAM Journal on Matrix Analysis and Applications*, vol. 31, no. 3, pp. 1235–1256, 2009.

[46] X. Su and T. M. Khoshgoftaar, "A survey of collaborative filtering techniques," *Advances in artificial intelligence*, vol. 2009, p. 4, 2009.

[47] Y. Hu, Y. Koren, and C. Volinsky, "Collaborative filtering for implicit feedback datasets," in *Proc. IEEE ICDM*. Ieee, 2008, pp. 263–272.

[48] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme, "BPR: Bayesian personalized ranking from implicit feedback," in *Proc. UAI*. AUAI Press, 2009, pp. 452–461.

[49] B. Liu, D. Niu, Z. Li, and H. V. Zhao, "Network latency prediction for personal devices: Distance-feature decomposition from 3d sampling," in *Computer Communications (INFOCOM), 2015 IEEE Conference on*. IEEE, 2015, pp. 307–315.

[50] Z. Zheng, Y. Zhang, and M. R. Lyu, "Investigating QoS of real-world web services," *IEEE Trans. Service Comput.*, vol. 7, no. 1, pp. 32–39, 2014.

[51] E. J. Candes and T. Tao, "Decoding by linear programming," *Information Theory, IEEE Transactions on*, vol. 51, no. 12, pp. 4203–4215, 2005.

[52] R. Chartrand and W. Yin, "Iteratively reweighted algorithms for compressive sensing," in *Acoustics, speech and signal processing, 2008. ICASSP 2008. IEEE international conference on*. IEEE, 2008, pp. 3869–3872.

[53] S. Ma, D. Goldfarb, and L. Chen, "Fixed point and bregman iterative methods for matrix rank minimization," *Mathematical Programming*, vol. 128, no. 1-2, pp. 321–353, 2011.

[54] K. Mohan and M. Fazel, "Reweighted nuclear norm minimization with application to system identification," in *American Control Conference (ACC), 2010*. IEEE, 2010, pp. 2953–2959.

[55] A. Mnih and R. Salakhutdinov, "Probabilistic matrix factorization," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2007, pp. 1257–1264.

[56] Q. Zheng and J. Lafferty, "Convergence analysis for rectangular matrix completion using burer-monteiro factorization and gradient descent," *arXiv preprint arXiv:1605.07051*, 2016.

[57] T. Zhao, Z. Wang, and H. Liu, "A nonconvex optimization framework for low rank matrix estimation," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2015, pp. 559–567.

[58] S. Tu, R. Boczar, M. Simchowitz, M. Soltanolkotabi, and B. Recht, "Low-rank solutions of linear matrix equations via procrustes flow," in *Proc. International Conference on Machine Learning*, 2016.

[59] L. Wang, X. Zhang, and Q. Gu, "A unified computational and statistical framework for nonconvex low-rank matrix estimation," in *Proc. International Conference on Artificial Intelligence and Statistics (AISTATS), available: arXiv:1610.05275*, 2017.

[60] D. Park, A. Kyrillidis, C. Caramanis, and S. Sanghavi, "Finding low-rank solutions via nonconvex matrix factorization, efficiently and provably," *SIAM Journal on Imaging Sciences*, vol. 11, no. 4, pp. 2165–2204, 2018.

[61] H. Robbins and S. Monro, "A stochastic approximation method," *The annals of mathematical statistics*, pp. 400–407, 1951.

[62] L. Bottou, "Stochastic gradient learning in neural networks," *Proceedings of Neuro-Nımes*, vol. 91, no. 8, 1991.

[63] A. Nemirovskii, D. B. Yudin, and E. R. Dawson, "Problem complexity and method efficiency in optimization," 1983.

[64] E. Moulines and F. R. Bach, "Non-asymptotic analysis of stochastic approximation algorithms for machine learning," in *Advances in Neural Information Processing Systems*, 2011, pp. 451–459.

[65] A. Nemirovski, A. Juditsky, G. Lan, and A. Shapiro, "Robust stochastic approximation approach to stochastic programming," *SIAM Journal on optimization*, vol. 19, no. 4, pp. 1574–1609, 2009.

[66] G. Lan, "An optimal method for stochastic composite optimization," *Mathematical Programming*, vol. 133, no. 1, pp. 365–397, 2012.

[67] J. Duchi and Y. Singer, "Efficient online and batch learning using forward backward splitting," *Journal of Machine Learning Research*, vol. 10, no. Dec, pp. 2899–2934, 2009.

[68] S. Ghadimi, G. Lan, and H. Zhang, "Mini-batch stochastic approximation methods for nonconvex stochastic composite optimization," *Mathematical Programming*, vol. 155, no. 1-2, pp. 267–305, 2016.

[69] P. Tseng, "On the rate of convergence of a partially asynchronous gradient projection algorithm," *SIAM Journal on Optimization*, vol. 1, no. 4, pp. 603–619, 1991.

[70] M. Razaviyayn, M. Hong, Z.-Q. Luo, and J.-S. Pang, "Parallel successive convex approximation for nonsmooth nonconvex optimization," in *Advances in Neural Information Processing Systems*, 2014, pp. 1440–1448.

[71] D. Davis, "The asynchronous palm algorithm for nonsmooth nonconvex problems," *arXiv preprint arXiv:1604.00526*, 2016.

[72] Y. Zhou, Y. Yu, W. Dai, Y. Liang, and E. Xing, "On convergence of model parallel proximal gradient algorithm for stale synchronous parallel system," in *Artificial Intelligence and Statistics*, 2016, pp. 713–722.

[73] J. Liu and S. J. Wright, "Asynchronous stochastic coordinate descent: Parallelism and convergence properties," *SIAM Journal on Optimization*, vol. 25, no. 1, pp. 351–376, 2015.

[74] Y. Xu and W. Yin, "Block stochastic gradient iteration for convex and nonconvex optimization," *SIAM Journal on Optimization*, vol. 25, no. 3, pp. 1686–1716, 2015.

[75] X. Lian, Y. Huang, Y. Li, and J. Liu, "Asynchronous parallel stochastic gradient for nonconvex optimization," in *Advances in Neural Information Processing Systems*, 2015, pp. 2737–2745.

[76] H. Mania, X. Pan, D. Papailiopoulos, B. Recht, K. Ramchandran, and M. I. Jordan, "Perturbed iterate analysis for asynchronous stochastic optimization," *SIAM Journal on Optimization*, vol. 27, no. 4, pp. 2202–2229, jan 2017. [Online]. Available: https://doi.org/10.1137%2F16m1057000

[77] R. Zhang and J. Kwok, "Asynchronous distributed admm for consensus optimization," in *International Conference on Machine Learning*, 2014, pp. 1701–1709.

[78] M. Hong, "A distributed, asynchronous and incremental algorithm for nonconvex optimization: An admm approach," *IEEE Transactions on Control of Network Systems*, vol. PP, no. 99, pp. 1–1, 2017.

[79] T.-H. Chang, M. Hong, W.-C. Liao, and X. Wang, "Asynchronous distributed admm for large-scale optimization – part i: Algorithm and convergence analysis," *IEEE Transactions on Signal Processing*, vol. 64, no. 12, pp. 3118–3130, 2016.

[80] T.-H. Chang, W.-C. Liao, M. Hong, and X. Wang, "Asynchronous distributed admm for large-scale optimization – part ii: Linear convergence analysis and numerical performance," *IEEE Transactions on Signal Processing*, vol. 64, no. 12, pp. 3131–3144, 2016.

[81] E. Wei and A. Ozdaglar, "On the o (1/k) convergence of asynchronous distributed alternating direction method of multipliers," *arXiv preprint arXiv:1307.8254*, 2013.

[82] J. F. Mota, J. M. Xavier, P. M. Aguiar, and M. Puschel, "D-admm: A communication-efficient distributed algorithm for separable optimization," *IEEE Transactions on Signal Processing*, vol. 61, no. 10, pp. 2718–2723, 2013.

[83] H. R. Feyzmahdavian, A. Aytekin, and M. Johansson, "An asynchronous mini-batch algorithm for regularized stochastic optimization," in *Decision and Control (CDC), 2015 IEEE 54th Annual Conference on*. IEEE, 2015, pp. 1384–1389.

[84] M. Li, D. G. Andersen, A. J. Smola, and K. Yu, "Communication efficient distributed machine learning with the parameter server," in *Advances in Neural Information Processing Systems*, 2014, pp. 19–27.

[85] D. Davis, B. Edmunds, and M. Udell, "The sound of apalm clapping: Faster nonsmooth nonconvex optimization with stochastic asynchronous palm," in *Advances in Neural Information Processing Systems*, 2016, pp. 226–234.

[86] K. Mohan and M. Fazel, "Iterative Reweighted Algorithms for Matrix Rank Minimization," *The Journal of Machine Learning Research (JMLR)*, vol. 13, no. 1, pp. 3441–3473, 2012.

[87] Y. Zhang and Z. Lu, "Penalty Decomposition Methods for Rank Minimization," in *Proc. Advances in Neural Information Processing Systems (NIPS)*, 2011.

[88] I. Daubechies, R. DeVore, M. Fornasier, and C. S. Güntürk, "Iteratively Reweighted Least Squares Minimization for Sparse Recovery," *Comm. Pure Appl. Math*, vol. 63, no. 1, pp. 1–38, 2010.

[89] M. S. Lobo, M. Fazel, and S. Boyd, "Portfolio Optimization with Linear and Fixed Transaction Costs," *Ann. Oper. Res.*, vol. 152, no. 1, pp. 341–365, 2007.

[90] E. J. Candès and T. Tao, "The Power of Convex Relaxation: Near-optimal Matrix Completion," *IEEE Trans. Info. Theory (TIT)*, vol. 56, no. 5, pp. 2053–2080, 2010.

[91] J. Xu, "Reweighted nuclear norm minimization for matrix completion," *Proceedings available online at https://webspace. utexas. edu/jx598/www/Reweighted. pdf*, 2011.

[92] V. De Silva and L.-H. Lim, "Tensor Rank and the Ill-Posedness of the Best Low-Rank Approximation Problem," *SIAM J. Matrix Anal. Appl.*, vol. 30, no. 3, pp. 1084–1127, 2008.

[93] S. Gandy, B. Recht, and I. Yamada, "Tensor Completion and Low-n-Rank Tensor Recovery via Convex Optimization," *Inverse Problems*, vol. 27, no. 2, 2011.

[94] B. Liu, D. Niu, Z. Li, and H. V. Zhao, "Network Latency Prediction for Personal Devices: Distance-Feature Decomposition from 3D Sampling," in *Proc. IEEE INFOCOM*, 2015.

[95] F. Dabek, R. Cox, F. Kaashoek, and R. Morris, "Vivaldi: A Decentralized Network Coordinate System," in *Proc. ACM SIGCOMM*, vol. 34, no. 4, 2004.

[96] I. Olkin, A. W. Marshall, and B. C. Arnold, *Inequalities: Theory of Majorization and Its Applications*, 2nd ed., ser. Springer Series in Statistics. New York: Springer, 2011.

[97] X. Chen, Z. Zheng, Q. Yu, and M. R. Lyu, "Web service recommendation via exploiting location and QoS information," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 7, pp. 1913–1924, 2014.

[98] B. Liu, D. Niu, Z. Li, and H. V. Zhao, "Network latency prediction for personal devices: Distance-feature decomposition from 3D sampling," in *Proc. IEEE INFOCOM*, 2015, pp. 307–315.

[99] R. Koenker, *Quantile regression*. Cambridge university press, 2005, no. 38.

[100] C. Chen, "A finite smoothing algorithm for quantile regression," *J. Comp. Graph. Stat.*, 2012.

[101] A. Beck, "On the convergence of alternating minimization for convex programming with applications to iteratively reweighted least squares and decomposition schemes," *SIAM Journal on Optimization*, vol. 25, no. 1, pp. 185–209, 2015.

[102] M. Razaviyayn, M. Hong, and Z.-Q. Luo, "A unified convergence analysis of block successive minimization methods for nonsmooth optimization," *SIAM J. Optim.*, vol. 23, no. 2, pp. 1126–1153, 2013.

[103] M. Hong, M. Razaviyayn, Z.-Q. Luo, and J.-S. Pang, "A unified algorithmic framework for block-structured optimization involving big data: With applications in machine learning and signal processing," *IEEE Signal Processing Magazine*, vol. 33, no. 1, pp. 57–77, 2016.

[104] Z. Zheng, H. Ma, M. R. Lyu, and I. King, "Collaborative web service qos prediction via neighborhood integrated matrix factorization," *IEEE Trans. Service Comput.*, vol. 6, no. 3, pp. 289–299, 2013.

[105] C. Eckart and G. Young, "The approximation of one matrix by another of lower rank," *Psychometrika*, vol. 1, no. 3, pp. 211–218, 1936.

[106] W. K. Newey and J. L. Powell, "Asymmetric least squares estimation and testing," *Econometrica*, pp. 819–847, 1987.

[107] A. P. Singh and G. J. Gordon, "A unified view of matrix factorization models," in *ECML PKDD*. Springer, 2008, pp. 358–373.

[108] P. Jain, P. Netrapalli, and S. Sanghavi, "Low-rank matrix completion using alternating minimization," in *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*. ACM, 2013, pp. 665–674.

[109] P. Jain, R. Meka, and I. S. Dhillon, "Guaranteed rank minimization via singular value projection," in *Advances in Neural Information Processing Systems*, 2010, pp. 937–945.

[110] G. Taylor, R. Burmeister, Z. Xu, B. Singh, A. Patel, and T. Goldstein, "Training neural networks without gradients: A scalable admm approach," in *International Conference on Machine Learning*, 2016, pp. 2722–2731.

[111] M. Hong, Z.-Q. Luo, and M. Razaviyayn, "Convergence analysis of alternating direction method of multipliers for a family of nonconvex problems," *SIAM Journal on Optimization*, vol. 26, no. 1, pp. 337–364, 2016.

[112] H. Li and Z. Lin, "Accelerated proximal gradient methods for nonconvex programming," in *Advances in neural information processing systems*, 2015, pp. 379–387.

[113] P. Moritz, R. Nishihara, S. Wang, A. Tumanov, R. Liaw, E. Liang, W. Paul, M. I. Jordan, and I. Stoica, "Ray: A distributed framework for emerging ai applications," *arXiv preprint arXiv:1712.05889*, 2017.

[114] H. Avron, A. Druinsky, and A. Gupta, "Revisiting asynchronous linear solvers: Provable convergence rate through randomization," *Journal of the ACM (JACM)*, vol. 62, no. 6, p. 51, 2015.

[115] D. Borthakur, "Hdfs architecture guide," *HADOOP APACHE PROJECT http://hadoop. apache. org/common/docs/current/hdfs design. pdf*, 2008.

Appendix A

Preliminary Lemmas for Chapter 7 and 8

A.1 Auxiliary Lemmas

Lemma A.1.1 ([68]). *For all* $\mathbf{y} \leftarrow \mathbf{prox}_{\eta h}(\mathbf{x} - \eta \mathbf{g})$*, we have:*

$$\langle \mathbf{g}, \mathbf{y} - \mathbf{x} \rangle + (h(\mathbf{y}) - h(\mathbf{x})) \leq -\frac{\|\mathbf{y} - \mathbf{x}\|_2^2}{\eta}. \tag{A.1}$$

Due to slightly different notations and definitions in [68], we provide a proof here for completeness. We refer readers to [68] for more details.

Proof. By the definition of proximal function, there exists a $\mathbf{p} \in \partial h(\mathbf{y})$ such that:

$$\begin{aligned} \langle \mathbf{g} + \frac{\mathbf{y} - \mathbf{x}}{\eta} + \mathbf{p}, \mathbf{x} - \mathbf{y} \rangle &\geq 0, \\ \langle \mathbf{g}, \mathbf{x} - \mathbf{y} \rangle &\geq \frac{1}{\eta} \langle \mathbf{y} - \mathbf{x}, \mathbf{y} - \mathbf{x} \rangle + \langle p, \mathbf{y} - \mathbf{x} \rangle \\ \langle g, \mathbf{x} - \mathbf{y} \rangle + (h(\mathbf{x}) - h(\mathbf{y})) &\geq \frac{1}{\eta} \|\mathbf{y} - \mathbf{x}\|_2^2, \end{aligned}$$

which proves the lemma. □

The following corollary is used for analyzing updates on a single block:

Corollary A.1.1. *For all* $y_j \leftarrow \mathbf{prox}_{\eta h_j}(x_j - \eta g_j)$*, we have:*

$$\langle g_j, y_j - x_j \rangle + (h_j(y_j) - h_j(x_j)) \leq -\frac{\|y_j - x_j\|_2^2}{\eta}. \tag{A.2}$$

Lemma A.1.2 ([68]). *For all $\mathbf{x}, \mathbf{g}, \mathbf{G} \in \mathbb{R}^d$, if $h : \mathbb{R}^d \to \mathbb{R}$ is a convex function, we have*

$$\|\mathbf{prox}_{\eta h}(\mathbf{x} - \eta\mathbf{G}) - \mathbf{prox}_{\eta h}(\mathbf{x} - \eta\mathbf{g})\| \leq \eta\|\mathbf{G} - \mathbf{g}\|. \tag{A.3}$$

Proof. Let $\mathbf{y}$ denote $\mathbf{prox}_{\eta h}(\mathbf{x} - \eta\mathbf{G})$ and $\mathbf{z}$ denote $\mathbf{prox}_{\eta h}(\mathbf{x} - \eta\mathbf{g})$. By definition of the proximal operator, for all $\mathbf{u} \in \mathbb{R}^d$, we have

$$\langle \mathbf{G} + \frac{\mathbf{y} - \mathbf{x}}{\eta} + \mathbf{p}, \mathbf{u} - \mathbf{y} \rangle \geq 0,$$
$$\langle \mathbf{g} + \frac{\mathbf{z} - \mathbf{x}}{\eta} + \mathbf{q}, \mathbf{u} - \mathbf{z} \rangle \geq 0,$$

where $p \in \partial h(\mathbf{y})$ and $q \in \partial h(\mathbf{z})$. Let $\mathbf{z}$ substitute $\mathbf{u}$ in the first inequality and $\mathbf{y}$ in the second one, we have

$$\langle \mathbf{G} + \frac{\mathbf{y} - \mathbf{x}}{\eta} + \mathbf{p}, \mathbf{z} - \mathbf{y} \rangle \geq 0,$$
$$\langle \mathbf{g} + \frac{\mathbf{z} - \mathbf{x}}{\eta} + \mathbf{q}, \mathbf{y} - \mathbf{z} \rangle \geq 0.$$

Then, we have

$$\langle \mathbf{G}, \mathbf{z} - \mathbf{y} \rangle \geq \langle \frac{\mathbf{y} - \mathbf{x}}{\eta}, \mathbf{y} - \mathbf{z} \rangle + \langle \mathbf{p}, \mathbf{y} - \mathbf{z} \rangle, \tag{A.4}$$
$$= \frac{1}{\eta}\langle \mathbf{y} - \mathbf{z}, \mathbf{y} - \mathbf{z} \rangle + \frac{1}{\eta}\langle \mathbf{z} - \mathbf{x}, \mathbf{y} - \mathbf{z} \rangle + \langle \mathbf{p}, \mathbf{y} - \mathbf{z} \rangle, \tag{A.5}$$
$$\geq \frac{\|\mathbf{y} - \mathbf{z}\|^2}{\eta} + \frac{1}{\eta}\langle \mathbf{z} - \mathbf{x}, \mathbf{y} - \mathbf{z} \rangle + h(\mathbf{y}) - h(\mathbf{z}), \tag{A.6}$$

and

$$\langle \mathbf{g}, \mathbf{y} - \mathbf{z} \rangle \geq \langle \frac{\mathbf{z} - \mathbf{x}}{\eta} + \mathbf{g}, \mathbf{z} - \mathbf{y} \rangle, \tag{A.7}$$
$$= \frac{1}{\eta}\langle \mathbf{z} - \mathbf{x}, \mathbf{z} - \mathbf{y} \rangle + \langle \mathbf{q}, \mathbf{z} - \mathbf{y} \rangle \tag{A.8}$$
$$\geq \frac{1}{\eta}\langle \mathbf{z} - \mathbf{x}, \mathbf{z} - \mathbf{y} \rangle + h(\mathbf{z}) - h(\mathbf{y}). \tag{A.9}$$

By adding (A.6) and (A.9), we obtain

$$\|\mathbf{G} - \mathbf{g}\|\|\mathbf{z} - \mathbf{y}\| \geq \langle \mathbf{G} - \mathbf{g}, \mathbf{z} - \mathbf{y} \rangle \geq \frac{1}{\eta}\|\mathbf{y} - \mathbf{z}\|^2,$$

which proves the lemma. □

Corollary A.1.2. *For all $x_j, g_j, G_j \in \mathbb{R}^{d_j}$, we have*

$$\|\mathbf{prox}_{\eta h_j}(x_j - \eta G_j) - \mathbf{prox}_{\eta h_j}(x_j - \eta g_j)\| \leq \eta\|G_j - g_j\|. \tag{A.10}$$

Lemma A.1.3 ([68]). *For any* $\mathbf{g}_1$ *and* $\mathbf{g}_2$*, we have*

$$\|P(\mathbf{x}, \mathbf{g}_1, \eta) - P(\mathbf{x}, \mathbf{g}_2, \eta)\| \leq \|\mathbf{g}_1 - \mathbf{g}_2\|. \tag{A.11}$$

Proof. It can be obtained by directly applying Lemma A.1.2 and the definition of gradient mapping. □

Corollary A.1.3. *Let* $P_j(\mathbf{x}, \mathbf{g}, \eta) := \frac{1}{\eta}(x_j - \mathbf{prox}_{\eta h_j}(x_j - \eta g_j))$*. Then, for any* G_j *and* g_j*, we have*

$$\|P_j(\mathbf{x}, \mathbf{G}, \eta) - P_j(\mathbf{x}, \mathbf{g}, \eta)\| \leq \|G_j - g_j\|. \tag{A.12}$$

Lemma A.1.4 ([25]). *Suppose we define* $\mathbf{y} = \mathbf{prox}_{\eta h}(\mathbf{x} - \eta \mathbf{g})$ *for some* $\mathbf{g}$*. Then for* $\mathbf{y}$*, the following inequality holds:*

$$\begin{aligned}\Psi(\mathbf{y}) \leq \Psi(\mathbf{z}) &+ \langle \mathbf{y} - \mathbf{z}, \nabla f(\mathbf{x}) - \mathbf{g} \rangle \\ &+ \left(\frac{L}{2} - \frac{1}{2\eta}\right)\|\mathbf{y} - \mathbf{x}\|^2 + \left(\frac{L}{2} + \frac{1}{2\eta}\right)\|\mathbf{z} - \mathbf{x}\|^2 - \frac{1}{2\eta}\|\mathbf{y} - \mathbf{z}\|^2,\end{aligned} \tag{A.13}$$

for all $\mathbf{z}$*.*

Corollary A.1.4. *Suppose we define* $y_j = \mathbf{prox}_{\eta h_j}(x_j - \eta g_j)$ *for some* g_j*, and the index* j *is chosen among* M *indices with uniform distribution. For other* $j' \neq j$*, we assume* $y_{j'} = x_{j'}$*. Then the following inequality holds:*

$$\begin{aligned}\Psi(\mathbf{y}) \leq \Psi(\mathbf{z}) &+ \langle \nabla_j f(\mathbf{x}) - g_j, y_j - z_j \rangle \\ &+ \left(\frac{L_j}{2} - \frac{1}{2\eta}\right)\|y_j - x_j\|^2 + \left(\frac{L_j}{2} + \frac{1}{2\eta}\right)\|z_j - x_j\|^2 - \frac{1}{2\eta}\|y_j - x_j\|^2.\end{aligned} \tag{A.14}$$

for all $\mathbf{z}$*.*

Proof. From the definition of proximal operator, we have

$$\begin{aligned}&h_j(y_j) + \langle g_j, y_j - x_j \rangle + \frac{1}{2\eta}\|y_j - x_j\|^2 + \frac{\eta}{2}\|g_j\|^2 \\ \leq\ &h_j(z_j) + \langle g_j, z_j - x_j \rangle + \frac{1}{2\eta}\|z_j - x_j\|^2 + \frac{\eta}{2}\|g_j\|^2 - \frac{1}{2\eta}\|y_j - z_j\|^2.\end{aligned}$$

By rearranging the above inequality, we have

$$h_j(y_j) + \langle g_j, y_j - z_j \rangle \leq h_j(z_j) + \frac{1}{2\eta}[\|z_j - x_j\|^2 - \|y_j - x_j\|^2 - \|y_j - z_j\|^2]. \tag{A.15}$$

Since f is L-Lipschitz, we have

$$f(\mathbf{y}) \leq f(\mathbf{x}) + \langle \nabla_j f(\mathbf{x}), y_j - x_j \rangle + \frac{L_j}{2}\|y_j - x_j\|^2,$$
$$f(\mathbf{x}) \leq f(\mathbf{z}) + \langle \nabla_j f(\mathbf{x}), x_j - z_j \rangle + \frac{L_j}{2}\|x_j - z_j\|^2.$$

Adding these two inequality we have

$$f(\mathbf{y}) \leq f(\mathbf{z}) + \langle \nabla_j f(\mathbf{x}), y_j - z_j \rangle + \frac{L}{2}[\|y_j - x_j\|^2 + \|z_j - x_j\|^2], \tag{A.16}$$

and therefore

$$\Psi(\mathbf{y}) \leq \Psi(\mathbf{z}) + \langle \nabla_j f(\mathbf{x}) - g_j, y_j - z_j \rangle + \left(\frac{L_j}{2} - \frac{1}{2\eta}\right)\|y_j - x_j\|^2 + \left(\frac{L_j}{2} + \frac{1}{2\eta}\right)\|z_j - x_j\|^2 - \frac{1}{2\eta}\|y_j - x_j\|^2.$$

□

Lemma A.1.5 (Young's Inequality)**.**

$$\langle \mathbf{a}, \mathbf{b} \rangle \leq \frac{1}{2\delta}\|\mathbf{a}\|^2 + \frac{\delta}{2}\|\mathbf{b}\|^2. \tag{A.17}$$