Effective and Efficient Learning at Scale

Adams Wei Yu

AUGUST 2019
CMU-ML-19-111

Machine Learning Department
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Jaime Carbonell, Co-Chair
Alexander Smola, Co-Chair (Amazon)
Ruslan Salakhutdinov
Quoc Le (Google)
Christopher Manning (Stanford)

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

This research was supported by a grant from the Boeing Company, a fellowship from the Nvidia Corporation, a fellowship from the Snap Inc, a fellowship from the Thomas and Stacey Siebel Foundation and a Carnegie Mellon University presidential fellowship.

Keywords: Large Scale Optmization, Deep Neural Networks, Machine Reading Comprehension, Natural Language Processing

To my beloved wife, Yanyu Long and son, Ziheng Yu.

Abstract

How to enable efficient and effective machine learning at scale has been a long-standing problem in modern artificial intelligence, which also motivates this thesis research. In particular, we aim at solving the following problems:

1. How to efficiently train a machine learning model?
2. How to speed up inference after the model is trained?
3. How to make the model generalize better?

We approach those problems from two perspectives: **models** and **algorithms**. On one hand, we design novel models that are intrinsically fast to train and/or test. On the other, we develop new algorithms with rapid convergence guarantee. Not surprisingly, the above three problem are not mutually exclusive and thus solving one of them might also benefit others. For example, 1) a new model that can enable parallel computation helps accelerate both training and inference; 2) a fast algorithm can save time for hyper-parameter tuning and/or make it affordable for training with more data, which in return boosts the generalization performance.

This thesis consists of two parts. The first part presents new machine learning models with a focus on sequential data such as natural language processing and question answering. Firstly, we propose a model, LSTM-Jump, that can skip unimportant information in text, mimicking the skimming behavior of human reading. Trained with an efficient reinforcement learning algorithm, this model can be several times faster than a vanilla LSTM in inference time. Then we introduce a text encoding model that totally discards recurrent networks, which thus fully supports parallel training and inference. Based on this technique, a new question-answering model, QANet, is proposed. Combined with data augmentation approach via back-translation, this model stays at the No.1 place in the competitive Stanford Question and Answer Dataset (SQuAD) from March to Aug 2018, while being times faster than the prevalent models. It was also the deepest neural network model for NLP when invented.

The second part proposes large scale learning algorithms with provable convergence guarantee. To enable fast training of neural networks, we propose a general gradient normalization algorithm for efficient deep networks training. This method can not only alleviate the gradient vanishing problem, but also regularize the model to achieve better generalization. When the amount of training data becomes huge, we need appeal to distributed computation. We are particularly interested in the ubiquitous problem, empirical risk minimization, and propose a few algorithms to attack the challenges posed by the distributed environment. We first show that a randomized primal-dual coordinate method DSPDC can achieve a linear convergence rate in the single-machine setting. Then by further leveraging the structural information of the problem, we propose an asynchronous algorithm DSCOVR, which enjoys a linear convergence rate as well on the distributed parameter server environment, by executing periodical variance reduction.

Acknowledgments

It has been a great journey for me in the past few years as a PhD student at CMU, filled with growing, learning and developing. I could not have asked for more.

The first person I am greatly indebted to along this journey is my advisor Jaime Carbonell, the nicest person I have ever met in my life. Jaime always gives me a lot of freedom in choosing research topics and publishing in different areas, which helps me become a well-rounded young researcher. As a guru in machine learning and natural language processing, he knows almost everything in these two areas and provides plenty of insightful advices for each topic I had worked on. Without his kind support and guidance, I could not have achieved what I have now, not even close. Jaime is also a perfect role model of diligence. He is the only professor I know who had worked at CMU for 40 years but never taken a sabbatical break. Working with such an intelligent and hard-working person makes me hardly slow down my pace in research.

I would also like to thank Alex Smola, my another advisor. Alex has tons of research ideas every day, a lot more than what I can digest, yet most of which would lead to great papers. Although it was a pity that Alex left CMU in the middle, his last wisdom proved to be very important for me – "Walk out of your comfort zone". I followed his suggestion and started to work on deep learning in 2016, which totally changed my research path.

I should thank Quoc Le as well, who recruited me to Google Brain as an intern, even when he knew I was completely a layman of deep learning. Quoc opens the door for me to neural networks, teaches me how to think out of the box, and shares with me uncountably many crazy ideas. I am really lucky to have him as a mentor and close friend in my career.

Thanks also goes to the other thesis committee members, Ruslan Salakhutdinov and Chris Manning, for their time and effort on giving me invaluable feedbacks. As a world-class expert, Russ always impresses me by his deep understanding of every aspect of neural networks. I really enjoy the discussion with him, whose output turns out to be part of this thesis. I still remember the questions raised by Chris when I was giving a talk in Stanford NLP group, which helped shape my oral presentation. The meeting in person after the talk was also enlightening. I was grateful that Chris shared his viewpoint on the future of NLP under the deep learning era.

I am fortunate enough to have worked with so many talents in optimization, which is a major part of this thesis. Firstly, I would thank Fatma Kılınç-Karzan, who mentored me on my first paper in CMU, which turned out to be an INFORMS data mining best student paper candidate. I need to thank Suvrit Sra, who taught a fantastic course Advanced Optimization and Randomized Methods, the course project of which turned out to be my second paper in CMU. The discussion and collaboration after he moved to MIT continued, which was enjoyable. I also enjoy the collaboration with Qihang Lin, one of the solidest optimization researchers I have ever met. I really appreciate Lin Xiao for hosting me in Microsoft Research Redmond for a fruitful summer internship. He set up a good example for me, teaching me

the importance of patience in doing solid research in optimization. I was impressed by the breadth of Yaoliang Yu's horizon, and the discussion with him was always enlightening.

I would not be able to grow to what I have been today without my fantastic collaborators, whom I would like to thank here as well: Maria Florina Balcan, Jaime Carbonell, Kai Chen, Weizhu Chen, David Dohan, Simon S. Du, Lei Huang, Fatma Kılınç-Karzan, Quoc Le, Hongrae Lee, Bo Li, Mu Li, Qihang Lin, Thang Luong, Wanli Ma, Mohammad Norouzi, Ruslan Salakhutdinov, Aarti Singh, Alex Smola, Suvrit Sra, Yining Wang, Lin Xiao, Tianbao Yang, Yaoliang Yu and Rui Zhao.

I am also grateful to Diane Stidle, Mallory Deptola, Alison Chiocchi and all other MLD stuff members for making our big ML family well-organized. I want to say sorry to Diane for always bringing her trouble but she is consistently patient and helpful. I apologize for forgetting someone who should be acknowledged.

Last but not the least, I need to thank my parents for their long-lasting support and help during my life. I am particularly thankful to my wife Yanyu Long. Yanyu, to maintain our family ship, you gave up the job in China, came with me to CMU, started from scratch to become a surgeon in the US and gave birth to our lovely boy in the meanwhile. You have faced much higher pressure and conquered way more challenges than I have, both physically and psychologically, with your bravery, courage and determination. You are simply my hero and this thesis is dedicated to you!

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background and Motivation

The past decade has witnessed the great success of machine learning, especially deep learning (also known as deep neural networks), in various areas, including image classification [61, 88], speech recognition [67] and recently natural language processing [42]. The revival of deep neural networks is mainly due to the following facts: 1) non-linear models, for example neural networks, can capture such a broad function class that it is more powerful in learning representation for a plethora of applications, compared to traditional linear models; 2) The recent boost of computational power, especially the wide use of GPU has made it feasible to train neural networks within reasonable time.

However, as the modern neural network structures turn deeper and more sophisticated, and the data scale grows larger, the training and inference time will become consistently longer, even in presence of the significant acceleration brought by hardware development. For example, 1) we still need a few days to train a 100-layer neural net for ImageNet classification with modern computing facilities and 2) the current machine reading comprehension algorithms are not fast enough to be deployed on a real-time service.

In other words, speed is always the bottleneck for the current machine/deep learning research, which motivates this Ph.D study. This thesis encompasses deep learning and large scale optimization, two important and closely related fields in modern artificial intelligence. In particular, it focuses on accelerating machine learning from two perspectives: 1) inventing novel models to circumvent the disadvantages of existing architectures and 2) developing new algorithms to address the long-standing optimization issues. While speedup is the main theme, we also propose techniques to improve the generalization of the models, which could not have been accomplished without the acceleration.

In the rest of this chapter, we will give a general overview of the problems considered in this thesis, followed by sketches of their solutions.

1.2 Part One: Efficient and Effective Deep Learning Models for Sequential Data Processing

The first part of the thesis focuses on fast models for sequential data, a ubiquitous data format in many applications, with a particular interest in applications on natural language processing (NLP) and questions answering (QA). As deep neural networks become prevalent models in modern natural language, or more broadly sequential data processing, this part aims to design efficient and effective neural architectures.

1.2.1 Fast Recurrent Model to Capture Important Information

Recurrent neural networks (RNNs) have proven to be powerful tools to model sequence data, such as videos, natural languages and time series, and hence serve as building blocks for varieties of neural network architectures. Promising as they sound, there is a long-standing issue for RNNs – They are hardly able to capture the long term dependency in the data due to the redundant and noisy nature of most sequential data. It is analogous to human reading – one might forget or misremember important contents in the previous chapters of a book as s/he moves on.

In Chapter 2, inspired by the skimming skills of humans, we propose a RNN model with "skipping" connections, hoping that it can mimic the skim behavior that useful information is stored while noisy information is skipped. Specifically, after processing a few elements of a sequence, the model will predict how many immediate inputs can be skipped and directly "jump" to the next informative input. The benefits are two-fold: keeping important tokens only and processing much fewer inputs to enable speedup. Empirical study showed that we can achieve 6 times speedup on reading comprehension tasks with higher accuracy, while the visualization of the result indicates the model can indeed capture the key information. This chapter is based on a previous joint work [210] with Hongrae Lee and Quoc Le.

1.2.2 Recurrence Free Model for Parallel Training and Inference

While the RNN model with skip connections proposed in the previous section can alleviate the long term dependency issue, the sequential nature of RNNs is another fundamental obstacle for fast training. In a nutshell, that RNNs have to process the input tokens one after another makes it infeasible to process all the tokens within a sequence in parallel. However, the advantages of modern hardwares (such as GPUs or TPUs) for deep learning, can be exploited only when the computation is parallelizable. So how to develop new models to fit this trait is the key for further acceleration.

In Chapter 3, inspired by the parallelism friendly nature of convolution neural networks (CNN) and self-attention mechanism, we propose to completely replace the recurrence with the combination of those two components. The basic idea is to use convolution to sketch the local dependency of adjacent tokens and apply the self-attention to capture the global interaction of the distant tokens. The final architecture is a vertical pyramid rather than the horizontal long chain (the traditional RNN). Such a structural change enables the parallel processing of all the tokens of a sequence, resulting in 13 and 9 times speedup for training and inference respectively.

Upon proposed, this model was the *deepest* neural architecture in the NLP domain, consisting of more than 130 layers and 3 times deeper than the previous record. The depth immediately enables better generalization of the model. Furthermore, combined with a novel back-translation data augmentation techniques, our model QANet were No.1 on the competitive Stanford Question Answering (SQuAD) reading comprehension task in terms of both speed and accuracy. This chapter is based on a previous joint work [212] with David Dohan, Thang Luong, Rui Zhao, Kai Chen, Mohammad Norouzi and Quoc Le.

1.3 Part Two: Efficient Algorithms for General Model Training

The second part of the thesis aims to algorithmically improve the convergence of training, with extensive theoretical analysis and empirical studies.

1.3.1 Layer-wise Normalized Gradient Method for Training Deep Neural Networks

No matter what kind of neural networks (RNN/CNN/Attention) we use and how innovative we are in designing the model, at the end of the day, we need to apply a certain optimization algorithm to find the model parameters by minimizing some objective function that measures the discrepancy between ground truth and the model prediction. Such a parameter-finding procedure is called model training where the optimization algorithm is the dominating factor for convergence. Among all the algorithmic challenges, we are particularly interested in addressing those in training *very deep* neural networks, say, with more than 100 layers, as it has been a common sense that the deeper the network is, the more expressive the model can be, while the harder the training is.

One prominent challenge for training deep networks is due to the *gradient vanishing* phenomenon. That is, the feedback signal (gradient of the discrepancy) from the last layer (prediction) becomes weaker very quickly as it flows all the way down to the first layer (input), such that the bottom layers can hardly get signals to update the parameters. In view of this problem, we propose a normalized gradient with adaptive step-size method in Chapter 4. The idea is to maintain the gradient signal to a constant magnitude (gradient normalization) at each layer and then adjust the parameter update rule adaptively with the re-scaled signal. We have shown that within the same training time on a 101-layer neural network (ResNet-101), our algorithm significantly outperforms the state-of-the-art training algorithms on an image classification task, indicating the fast convergence of our method. This chapter is based on the previous joint work [211] with Lei Huang, Qihang Lin, Ruslan Salakhutdinov and Jaime Carbonell.

1.3.2 Distributed Optimization on Parameter Servers

Machine learning is always hungry for data and in most scenarios, the volume of training data would be so huge that it could no longer be stored in a single machine, where distributed storage

and computation are needed. A popular platform for this setting is the *Parameter Server* framework where the server maintains the global parameters and coordinates simultaneous updates of them at different data workers, while each data worker stores a portion of the training data and only communicates with the server. Such a dedicated system may significantly increase our capability of learning from big data, but at the same time, poses at least the following two new challenges for developing efficient distributed algorithms:

- Communication Efficiency: During the whole process of computation with this framework, write and read operations on the parameter server is extremely frequent, which results in significant communication overhead in the network. However, network bandwidth is much smaller than memory bandwidth and thus heavy communication cost would become an obstacle to the efficiency of the algorithm. So designing a communication efficient strategy is the key to improve the overall performance.
- Asynchrony: The computational speed of different data workers may vary a lot, which makes the slowest machine the bottleneck of any synchronous algorithm. Besides, in real distributed computing environment, the phenomenon of delay is inevitable and sometimes unpredictable, which multiple factors could contribute to, such as the CPU speed, I/O of disk, and network throughput. Therefore, to fully exploit the computation of all the data workers rather than spending most of time waiting, it is necessary but also challenging to develop asynchronous algorithms.

In Chapter 5, we attack those challenges starting from the simplest case, where there is only one machine to store data and update the parameters. We develop a new algorithm for the empirical risk minimization (ERM) problem, a generic form that can be instantiated as many machine learning problems, such as support vector machine, linear regression and logistic regression.

We reformulate the ERM as a bilinear saddle point problem, and propose a doubly stochastic primal-dual coordinate (DSPDC) optimization algorithm. In each iteration, our method randomly samples a block of coordinates of the primal and dual solutions to update. The linear convergence of our method could be established in terms of 1) the distance from the current iterate to the optimal solution and 2) the primal-dual objective gap. We show that the proposed method has a lower overall complexity than existing coordinate methods when either the data matrix has a factorized structure or the proximal mapping on each block is computationally expensive, e.g., involving an eigenvalue decomposition. The efficiency of DSPDC is confirmed by empirical studies on several real applications, such as the multi-task large margin nearest neighbor problem. This chapter is based on a previous joint work [209] with Qihang Lin and Tianbao Yang.

The study in Chapter 5 endows us with an efficient algorithm to deal with factorized data, which can be further leveraged in the distributed computation. In Chapter 6, we continue our studies on ERM in the real distributed computing environment, and propose a family of randomized primal-dual block coordinate algorithms that are especially suitable for asynchronous implementation with parameter servers. In particular, we work with the saddle-point formulation of such problems which allows simultaneous data and model partitioning, and exploit its structure by a new algorithms called doubly stochastic coordinate optimization with variance reduction (DSCOVR). Compared with other first-order distributed algorithms, we show that DSCOVR may require less amount of overall computation and communication, and less or no synchroniza-

tion. We discuss the implementation details of the DSCOVR algorithms, and present numerical experiments on an industrial distributed computing system. This chapter is based on a previous joint work [201] with Lin Xiao, Qihang Lin and Weizhu Chen.

1.4 Excluded Research

Besides my main stream research outlined above, I also have broad interests in designing and analyzing algorithms for various problems. The following works conducted during my Ph.D. study are excluded from the main content to keep this thesis succinct.

- Work on an accelerated perceptron algorithm for fast binary classification in [207].
- Work on a generalized condition gradient algorithm for structured matrix rank minimization problem [208].
- Work on Adadelay, an delay-adaptive stochastic distributed optimization algorithm [179].
- Work on the gap-dependency analysis of the noisy power method [13].
- Work on subset selection algorithm for linear regression under measurement constraints [189].
- Work on orthogonal weight normalization for deep neural network training [73].

Part I

Efficient and Effective Models for Sequential Data Processing

Chapter 2

Learning to Skip Unimportant Information in Sequential Data

2.1 Introduction

The last few years have seen much success of applying neural networks to many important applications in natural language processing, e.g., part-of-speech tagging, chunking, named entity recognition [30], sentiment analysis [176, 177], document classification [34, 82, 90, 218], machine translation [11, 81, 162, 181, 198], conversational/dialogue modeling [170, 178, 184], document summarization [123, 157], parsing [6] and automatic question answering (Q&A) [63, 95, 164, 182, 187, 190, 194, 203]. An important characteristic of all these models is that they read all the text available to them. While it is essential for certain applications, such as machine translation, this characteristic also makes it slow to apply these models to scenarios that have long input text, such as document classification or automatic Q&A. However, the fact that texts are usually written with redundancy inspires us to think about the possibility of reading selectively.

In this chapter, we consider the problem of understanding documents with partial reading, and propose a modification to the basic neural architectures that allows them to read input text with skipping. The main benefit of this approach is faster inference because it skips irrelevant information. An unexpected benefit of this approach is that it also helps the models generalize better.

In our approach, the model is a recurrent network, which learns to predict the number of jumping steps after it reads one or several input tokens. Such a discrete model is therefore not fully differentiable, but it can be trained by a standard policy gradient algorithm, where the reward can be the accuracy or its proxy during training.

In our experiments, we use the basic LSTM recurrent networks [69] as the base model and benchmark the proposed algorithm on a range of document classification or reading comprehension tasks, using various datasets such as Rotten Tomatoes [137], IMDB [111], AG News [218] and Children's Book Test [65]. We find that the proposed approach of selective reading speeds up the base model by two to six times. Surprisingly, we also observe our model beats the standard LSTM in terms of accuracy.

In summary, the main contribution of our work is to design an architecture that learns to skim

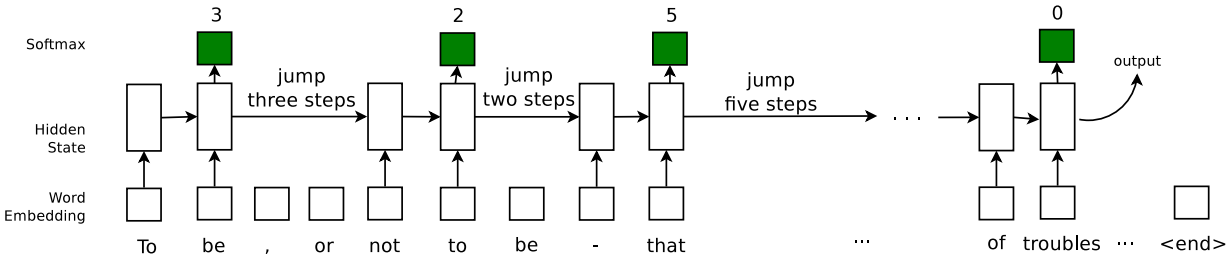

Figure 2.1: A synthetic example of the proposed model to process a text document. In this example, the maximum size of jump K is 5, the number of tokens read before a jump R is 2 and the number of jumps allowed N is 10. The green softmax are for jumping predictions. The processing stops if a) the jumping softmax predicts a 0 or b) the jump times exceeds N or c) the network processed the last token. We only show the case a) in this figure.

text and show that it is both faster and more accurate in practical applications of text processing. Our model is simple and flexible enough that we anticipate it would be able to incorporate to recurrent nets with more sophisticated structures to achieve even better performance in the future.

2.2 Related Work

Closely related to our work is the idea of learning visual attention with neural networks [9, 121, 165], where a recurrent model is used to combine visual evidence at multiple fixations processed by a convolutional neural network. Similar to our approach, the model is trained end-to-end using the REINFORCE algorithm [196]. However, a major difference between those work and ours is that we have to sample from discrete jumping distribution, while they can sample from continuous distribution such as Gaussian. The difference is mainly due to the inborn characteristics of text and image. In fact, as pointed out by Mnih et al. [121], it was difficult to learn policies over more than 25 possible discrete locations.

This idea has recently been explored in the context of natural language processing applications, where the main goal is to filter irrelevant content using a small network [25]. Perhaps the most closely related to our work is the concurrent work on learning to reason with reinforcement learning [172]. The key difference between our work and Shen et al. [172] is that they focus on early stopping after multiple pass of data to ensure accuracy whereas our method focuses on selective reading with single pass to enable fast processing.

The concept of "hard" attention has also been used successfully in the context of making neural network predictions more interpretable [96]. The key difference between our work and Lei et al. [96]'s method is that our method optimizes for faster inference, and is more dynamic in its jumping. Likewise is the difference between our approach and the "soft" attention approach by [11]. Recently, [56] investigate how machine can fixate and skip words, focusing on the comparison between the behavior of machine and human, while our goal is to make reading faster. They model the probability that each single word should be read in an unsupervised way while ours directly model the probability of how many words should be skipped with supervised learning.

Our method belongs to adaptive computation of neural networks, whose idea is recently explored by [54, 76], where different amount of computations are allocated dynamically per time step. The main difference between our method and Graves, Jernite et al.'s methods is that our method can set the amount of computation to be exactly zero for many steps, thereby achieving faster scanning over texts. Even though our method requires policy gradient methods to train, which is a disadvantage compared to [54, 76], we do not find training with policy gradient methods problematic in our experiments.

At the high-level, our model can be viewed as a simplified trainable Turing machine, where the controller can move on the input tape. It is therefore related to the prior work on Neural Turing Machines [55] and especially its RL version [216]. Compared to [216], the output tape in our method is more simple and reward signals in our problems are less sparse, which explains why our model is easy to train. It is worth noting that Zaremba and Sutskever report difficulty in using policy gradients to train their model.

Our method, by skipping irrelevant content, shortens the length of recurrent networks, thereby addressing the vanishing or exploding gradients in them [70]. The baseline method itself, Long Short Term Memory [69], belongs to the same category of methods. In this category, there are several recent methods that try to achieve the same goal, such as having recurrent networks that operate in different frequency [87] or is organized in a hierarchical fashion [21, 28].

Lastly, we should point out that we are among the recent efforts that deploy reinforcement learning to the field of natural language processing, some of which have achieved encouraging results in the realm of such as neural symbolic machine [99], machine reasoning [172] and sequence generation [148].

2.3 Methodology

In this section, we introduce the proposed model named LSTM-Jump. We first describe its main structure, followed by the difficulty of estimating part of the model parameters because of non-differentiability. To address this issue, we appeal to a reinforcement learning formulation and adopt a policy gradient method.

2.3.1 Model Overview

The main architecture of the proposed model is shown in Figure 2.1, which is based on an LSTM recurrent neural network. Before training, the number of jumps allowed N, the number of tokens read between every two jumps R and the maximum size of jumping K are chosen ahead of time. While K is a fixed parameter of the model, N and R are hyperparameters that can vary between training and testing. Also, throughout the paper, we would use $d_{1:p}$ to denote a sequence $d_1, d_2, ..., d_p$.

In the following, we describe in detail how the model operates when processing text. Given a training example $x_{1:T}$, the recurrent network will read the embedding of the first R tokens $x_{1:R}$ and output the hidden state. Then this state is used to compute the jumping softmax that determines a distribution over the jumping steps between 1 and K. The model then samples from this distribution a jumping step, which is used to decide the next token to be read into the model.

Let κ be the sampled value, then the next starting token is $x_{R+\kappa}$. Such process continues until either

a) the jump softmax samples a 0; or

b) the number of jumps exceeds N; or

c) the model reaches the last token x_T.

After stopping, as the output, the latest hidden state is further used for predicting desired targets. How to leverage the hidden state depends on the specifics of the task at hand. For example, for classification problems in Section 2.4.1, 2.4.2 and 2.4.3, it is directly applied to produce a softmax for classification, while in automatic Q&A problem of Section 2.4.4, it is used to compute the correlation with the candidate answers in order to select the best one. Figure 2.1 gives an example with $K = 5$, $R = 2$ and $N = 10$ terminating on condition a).

2.3.2 Training with REINFORCE

Our goal for training is to estimate the parameters of LSTM and possibly word embedding, which are denoted as θ_m, together with the jumping action parameters θ_a. Once obtained, they can be used for inference.

The estimation of θ_m is straightforward in the tasks that can be reduced as classification problems (which is essentially what our experiments cover), as the cross entropy objective $J_1(\theta_m)$ is differentiable over θ_m that we can directly apply backpropagation to minimize.

However, the nature of discrete jumping decisions made at every step makes it difficult to estimate θ_a, as cross entropy is no longer differentiable over θ_a. Therefore, we formulate it as a reinforcement learning problem and apply policy gradient method to train the model. Specifically, we need to maximize a reward function over θ_a which can be constructed as follows.

Let $j_{1:N}$ be the jumping action sequence during the training with an example $x_{1:T}$. Suppose h_i is a hidden state of the LSTM right before the i-th jump j_i,[1] then it is a function of $j_{1:i-1}$ and thus can be denoted as $h_i(j_{1:i-1})$. Now the jump is attained by sampling from the multinomial distribution $p(j_i|h_i(j_{1:i-1}); \theta_a)$, which is determined by the jump softmax. We can receive a reward R after processing $x_{1:T}$ under the current jumping strategy.[2] The reward should be positive if the output is favorable or non-positive otherwise. In our experiments, we choose

$$R = \begin{cases} 1 & \text{if prediction correct;} \\ -1 & \text{otherwise.} \end{cases}$$

Then the objective function of θ_a we want to maximize is the expected reward under the distribution defined by the current jumping policy, i.e.,

$$J_2(\theta_a) = \mathbb{E}_{p(j_{1:N};\theta_a)}[R]. \tag{2.3.1}$$

where $p(j_{1:N}; \theta_a) = \prod_i p(j_{1:i}|h_i(j_{1:i-1}); \theta_a)$.

[1] The i-th jumping step is usually *not* x_i.

[2] In the general case, one may receive (discounted) intermediate rewards after each jump. But in our case, we only consider final reward. It is equivalent to a special case that all intermediate rewards are identical and without discount.

Optimizing this objective numerically requires computing its gradient, whose exact value is intractable to obtain as the expectation is over high dimensional interaction sequences. By running S examples, an approximated gradient can be computed by the following REINFORCE algorithm [196]:

$$\begin{aligned}\nabla_{\theta_a} J_2(\theta_a) =& \sum_{i=1}^{N} \mathbb{E}_{p(j_{1:N};\theta_a)}[\nabla_{\theta_a} \log p(j_{1:i}|h_i;\theta_a)R] \\ \approx& \frac{1}{S}\sum_{s=1}^{S}\sum_{i=1}^{N}[\nabla_{\theta_a} \log p(j^s_{1:i}|h^s_i;\theta_a)R^s]\end{aligned}$$

where the superscript s denotes a quantity belonging to the s-th example. Now the term $\nabla_{\theta_a} \log p(j_{1:i}|h_i;\theta_a)$ can be computed by standard backpropagation.

Although the above estimation of $\nabla_{\theta_a} J_2(\theta_a)$ is unbiased, it may have very high variance. One widely used remedy to reduce the variance is to subtract a *baseline* value b^s_i from the reward R^s, such that the approximated gradient becomes

$$\nabla_{\theta_a} J_2(\theta_a) \approx \frac{1}{S}\sum_{s=1}^{S}\sum_{i=1}^{N}[\nabla_{\theta_a} \log p(j^s_{1:i}|h^s_i;\theta)(R^s - b^s_i)]$$

It is shown [196, 216] that any number b^s_i will yield an unbiased estimation. Here, we adopt the strategy of Mnih et al. [121] that $b^s_i = w_b h^s_i + c_b$ and the parameter $\theta_b = \{w_b, c_b\}$ is learned by minimizing $(R^s - b^s_i)^2$. Now the final objective to minimize is

$$J(\theta_m, \theta_a, \theta_b) = J_1(\theta_m) - J_2(\theta_a) + \sum_{s=1}^{S}\sum_{i=1}^{N}(R^s - b^s_i)^2,$$

which is fully differentiable and can be solved by standard backpropagation.

2.3.3 Inference

During inference, we can either use sampling or greedy evaluation by selecting the most probable jumping step suggested by the jump softmax and follow that path. In the our experiments, we will adopt the sampling scheme.

2.4 Experimental Results

In this section, we present our empirical studies to understand the efficiency of the proposed model in reading text. The tasks under experimentation are: synthetic number prediction, sentiment analysis, news topic classification and automatic question answering. Those, except the first one, are representative tasks in text reading involving different sizes of datasets and various levels of text processing, from character to word and to sentence. Table 2.1 summarizes the statistics of the dataset in our experiments.

Task	Dataset	Level	Vocab	AvgLen	#train	#valid	#test	#class
Number Prediction	synthetic	word	100	100 words	1M	10K	10K	100
Sentiment Analysis	Rotten Tomatoes	word	18,764	22 words	8,835	1,079	1,030	2
Sentiment Analysis	IMDB	word	112,540	241 words	21,143	3,857	25,000	2
News Classification	AG	character	70	200 characters	101,851	18,149	7,600	4
Q/A	Children Book Test-NE	sentence	53,063	20 sentences	108,719	2,000	2,500	10
Q/A	Children Book Test-CN	sentence	53,185	20 sentences	120,769	2,000	2,500	10

Table 2.1: Task and dataset statistics.

To exclude the potential impact of advanced models, we restrict our comparison between the vanilla LSTM [69] and our model, which is referred to as LSTM-Jump. In a nutshell, we show that, while achieving the same or even better testing accuracy, our model is up to 6 times and 66 times faster than the baseline LSTM model in real and synthetic datasets, respectively, as we are able to selectively skip a large fraction of text.

In fact, the proposed model can be readily extended to other recurrent neural networks with sophisticated mechanisms such as attention and/or hierarchical structure to achieve higher accuracy than those presented below. However, this is orthogonal to the main focus of this work and would be left as an interesting future work.

General Experiment Settings We use the Adam optimizer [83] with a learning rate of 0.001 in all experiments. We also apply gradient clipping to all the trainable variables with the threshold of 1.0. The dropout rate between the LSTM layers is 0.2 and the embedding dropout rate is 0.1. We repeat the notations N, K, R defined previously in Table 2.2, so readers can easily refer to when looking at Tables 2.4,2.5,2.6 and 2.7. While K is fixed during both training and testing, we would fix R and N at training but vary their values during test to see the impact of parameter changes. Note that N is essentially a constraint which can be relaxed. Yet we prefer to enforce this constraint here to let the model learn to read fewer tokens. Finally, the reported test time is measured by running one pass of the whole test set instance by instance, and the speedup is over the base LSTM model. The code is written with TensorFlow.[3]

Notation	Meaning
N	number of jumps allowed
K	maximum size of jumping
R	number of tokens read before a jump

Table 2.2: Notations referred to in experiments.

2.4.1 Number Prediction with a Synthetic Dataset

We first test whether LSTM-Jump is indeed able to learn how to jump if a *very clear* jumping signal is given in the text. The input of the task is a sequence of L positive integers $x_{0:T-1}$

[3] `https://www.tensorflow.org/`

and the output is simply x_{x_0}. That is, the output is chosen from the input sequence, with index determined by x_0 . Here are two examples to illustrate the idea:

$$\text{input1} : \underline{4}, 5, 1, 7, \underline{6}, 2. \quad \text{output1} : 6$$
$$\text{input2} : \underline{2}, 4, \underline{9}, 4, 5, 6. \quad \text{output2} : 9$$

One can see that x_0 is essentially the oracle jumping signal, i.e. the indicator of how many steps the reading should jump to get the exact output and obviously, the remaining number of the sequence are useless. After reading the first token, a "smart" network should be able to learn from the training examples to jump to the output position, skipping the rest.

We generate 1 million training and 10,000 validation examples with the rule above, each with sequence length $T = 100$. We also impose $1 \leq x_0 < T$ to ensure the index is valid. We find that directly training the LSTM-Jump with full sequence is unlikely to converge, therefore, we adopt a curriculum training scheme. More specifically, we generate sequences with lengths $\{10, 20, 30, 40, 50, 60, 70, 80, 90, 100\}$ and train the model starting from the shortest. Whenever the training accuracy reaches a threshold, we shift to longer sequences. We also train an LSTM with the same curriculum training scheme. The training stops when the validation accuracy is larger than 98%. We choose such stopping criterion simply because it is the highest that both models can achieve.[4] All the networks are single layered, with hidden size 512, embedding size 32 and batch size 100. During testing, we generate sequences of lengths 10, 100 and 1000 with the same rule, each having 10,000 examples. As the training size is large enough, we do not have to worry about overfitting so dropout is not applied. In fact, we find that the training, validation and testing accuracies are almost the same.

The results of LSTM and our method, LSTM-Jump, are shown in Table 2.3. The first observation is that LSTM-Jump is faster than LSTM; the longer the sequence is, the more significant speed-up LSTM-Jump can gain. This is because the well-trained LSTM-Jump is aware of the jumping signal at the first token and hence can directly jump to the output position to make prediction, while LSTM is agnostic to the signal and has to read the whole sequence. As a result, the reading speed of LSTM-Jump is hardly affected by the length of sequence, but that of LSTM is linear with respect to length. Besides, LSTM-Jump also outperforms LSTM in terms of test accuracy under all cases. This is not surprising either, as LSTM has to read a large amount of tokens that are potentially not helpful and could interfere with the prediction. In summary, the results indicate LSTM-Jump is able to learn to jump if the signal is clear.

2.4.2 Word Level Sentiment Analysis with Rotten Tomatoes and IMDB datasets

As LSTM-Jump has shown great speedups in the synthetic dataset, we would like to understand whether it could carry this benefit to real-world data, where "jumping" signal is not explicit. So in this section, we conduct sentiment analysis on two movie review datasets, both containing equal numbers of positive and negative reviews.

The first dataset is Rotten Tomatoes, which contains 10,662 documents. Since there is not a standard split, we randomly select around 80% for training, 10% for validation, and 10% for

[4] In fact, our model can get higher but we stick to 98% for ease of comparison.

Seq length	LSTM-Jump	LSTM	Speedup
Test accuracy			
10	**98%**	96%	n/a
100	**98%**	96%	n/a
1000	**90%**	80%	n/a
Test time (Avg tokens read)			
10	**13.5s (2.1)**	18.9s (10)	1.40x
100	**13.9s (2.2)**	120.4s (100)	8.66x
1000	**18.9s (3.0)**	1250s (1000)	**66.14x**

Table 2.3: Testing accuracy and time of synthetic number prediction problem. The jumping level is number.

testing. The average and maximum lengths of the reviews are 22 and 56 words respectively, and we pad each of them to 60. We choose the pre-trained word2vec embeddings[5] [120] as our fixed word embedding that we do not update this matrix during training. Both LSTM-Jump and LSTM contain 2 layers, 256 hidden units and the batch size is 100. As the amount of training data is small, we slightly augment the data by sampling a continuous 50-word sequence in each padded reviews as one training sample. During training, we enforce LSTM-Jump to read 8 tokens before a jump ($R = 8$), and the maximum skipping tokens per jump is 10 ($K = 10$), while the number of jumps allowed is 3 ($N = 3$).

The testing result is reported in Table 2.4. In a nutshell, LSTM-Jump is always faster than LSTM under different combinations of R and N. At the same time, the accuracy is on par with that of LSTM. In particular, the combination of $(R, N) = (7, 4)$ even achieves slightly better accuracy than LSTM while having a 1.5x speedup.

Model	(R, N)	Accuracy	Time	Speedup
LSTM-Jump	(9, 2)	0.783	**6.3s**	**1.98x**
	(8, 3)	0.789	7.3s	1.71x
	(7, 4)	**0.793**	8.1s	1.54x
LSTM	n/a	0.791	12.5s	1x

Table 2.4: Testing time and accuracy on the Rotten Tomatoes review classification dataset. The maximum size of jumping K is set to 10 for all the settings. The jumping level is word.

The second dataset is IMDB [111],[6] which contains 25,000 training and 25,000 testing movie reviews, where the average length of text is 240 words, much longer than that of Rotten Tomatoes. We randomly set aside about 15% of training data as validation set. Both LSTM-Jump and LSTM has one layer and 128 hidden units, and the batch size is 50. Again, we use pretrained word2vec embeddings as initialization but they are updated during training. We either pad a short sequence to 400 words or randomly select a 400-word segment from a long sequence as a training example. During training, we set $R = 20, K = 40$ and $N = 5$.

[5] `https://code.google.com/archive/p/word2vec/`
[6] `http://ai.Stanford.edu/amaas/data/sentiment/index.html`

Model	(R, N)	Accuracy	Time	Speedup
LSTM-Jump	(80, 8)	**0.894**	769s	1.62x
	(80, 3)	0.892	764s	1.63x
	(70, 3)	0.889	673s	1.85x
	(50, 2)	0.887	585s	2.12x
	(100, 1)	0.880	**489s**	**2.54x**
LSTM	n/a	0.891	1243s	1x

Table 2.5: Testing time and accuracy on the IMDB sentiment analysis dataset. The maximum size of jumping K is set to 40 for all the settings. The jumping level is word.

As Table 2.5 shows, the result exhibits a similar trend as found in Rotten Tomatoes that LSTM-Jump is uniformly faster than LSTM under many settings. The various (R, N) combinations again demonstrate the trade-off between efficiency and accuracy. If one cares more about accuracy, then allowing LSTM-Jump to read and jump more times is a good choice. Otherwise, shrinking either one would bring a significant speedup though at the price of losing some accuracy. Nevertheless, the configuration with the highest accuracy still enjoys a 1.6x speedup compared to LSTM. With a slight loss of accuracy, LSTM-Jump can be 2.5x faster .

2.4.3 Character Level News Article Classification with AG dataset

We now present results on testing the character level jumping with a news article classification problem. The dataset contains four classes of topics (World, Sports, Business, Sci/Tech) from the AG's news corpus,[7] a collection of more than 1 million news articles. The data we use is the subset constructed by Zhang et al. [218] for classification with character-level convolutional networks. There are 30,000 training and 1,900 testing examples for each class respectively, where 15% of training data is set aside as validation. The non-space alphabet under use are:

```
abcdefghijklmnopqrstuvwxyz0123456
789-,;.!?:/\|_@#$%&*~`'+-=<>()[]{}
```

Since the vocabulary size is small, we choose 16 as the embedding size. The initialized entries of the embedding matrix are drawn from a uniform distribution in $[-0.25, 0.25]$, which are progressively updated during training. Both LSTM-Jump and LSTM have 1 layer and 64 hidden units and the batch sizes are 20 and 100 respectively. The training sequence is again of length 400 that it is either padded from a short sequence or sampled from a long one. During training, we set $R = 30, K = 40$ and $N = 5$.

The result is summarized in Table 2.6. It is interesting to see that even with skipping, LSTM-Jump is not always faster than LSTM. This is mainly due to the fact that the embedding size and hidden layer are both much smaller than those used previously, and accordingly the processing of a token is much faster. In that case, other computation overhead such as calculating and sampling from the jump softmax might become a dominating factor of efficiency. By this cross-task comparison, we can see that the larger the hidden unit size of recurrent neural network and

[7] http://www.di.unipi.it/~gulli/AG_corpus_of_news_articles.html

the embedding are, the more speedup LSTM-Jump can gain, which is also confirmed by the task below.

Model	(R, N)	Accuracy	Time	Speedup
LSTM-Jump	(50, 5)	0.854	102s	0.80x
	(40, 6)	0.874	98.1s	0.83x
	(40, 5)	0.889	83.0s	0.98x
	(30, 5)	0.885	**63.6s**	**1.28x**
	(30, 6)	**0.893**	74.2s	1.10x
LSTM	n/a	0.881	81.7s	1x

Table 2.6: Testing time and accuracy on the AG news classification dataset. The maximum size of jumping K is set to 40 for all the settings. The jumping level is character.

2.4.4 Sentence Level Automatic Question Answering with Children's Book Test dataset

The last task is automatic question answering, in which we aim to test the sentence level skimming of LSTM-Jump. We benchmark on the data set Children's Book Test (CBT) [65].[8] In each document, there are 20 contiguous sentences (context) extracted from a children's book followed by a query sentence. A word of the query is deleted and the task is to select the best fit for this position from 10 candidates. Originally, there are four types of tasks according to the part of speech of the missing word, from which, we choose the most difficult two, i.e., the name entity (NE) and common noun (CN) as our focus, since simple language models can already achieve human-level performance for the other two types .

The models, LSTM or LSTM-Jump, firstly read the whole query, then the context sentences and finally output the predicted word. While LSTM reads everything, our jumping model would decide how many context sentences should skip after reading one sentence. Whenever a model finishes reading, the context and query are encoded in its hidden state $h_{\mathbf{O}}$, and the best answer from the candidate words has the same index that maximizes the following:

$$\text{softmax}(CWh_{\mathbf{O}}) \in \mathbb{R}^{10},$$

where $C \in \mathbb{R}^{10\times d}$ is the word embedding matrix of the 10 candidates and $W \in \mathbb{R}^{d\times\text{hidden\_size}}$ is a trainable weight variable. Using such bilinear form to select answer basically follows the idea of Chen et al. [22], as it is shown to have good performance. The task is now distilled to a classification problem of 10 classes.

We either truncate or pad each context sentence, such that they all have length 20. The same preprocessing is applied to the query sentences except that the length is set as 30. For both models, the number of layers is 2, the number of hidden units is 256 and the batch size is 32. Pretrained word2vec embeddings are again used and they are not adjusted during training. The maximum number of context sentences LSTM-Jump can skip per time is $K = 5$ while the

[8] `http://www.thespermwhale.com/jaseweston/babi/CBTest.tgz`

number of total jumping is limited to $N = 5$. We let the model jump after reading every sentence, so $R = 1$ (20 words).

The result is reported in Table 2.7. The performance of LSTM-Jump is superior to LSTM in terms of both accuracy and efficiency under all settings in our experiments. In particular, the fastest LSTM-Jump configuration achieves a remarkable 6x speedup over LSTM, while also having respectively 1.4% and 4.4% higher accuracy in Children's Book Test - Named Entity and Children's Book Test - Common Noun.

Model	(R, N)	Accuracy	Time	Speedup
Children's Book Test - Named Entity				
LSTM-Jump	(1, 5)	**0.468**	40.9s	3.04x
	(1, 3)	0.464	30.3s	4.11x
	(1, 1)	0.452	**19.9s**	**6.26x**
LSTM	n/a	0.438	124.5s	1x
Children's Book Test - Common Noun				
LSTM-Jump	(1, 5)	0.493	39.3s	3.09x
	(1, 3)	0.487	29.7s	4.09x
	(1, 1)	**0.497**	**19.8s**	**6.14x**
LSTM	n/a	0.453	121.5s	1x

Table 2.7: Testing time and accuracy on the Children's Book Test dataset. The maximum size of jumping K is set to 5 for all the settings. The jumping level is sentence.

The dominant performance of LSTM-Jump over LSTM might be interpreted as follows. After reading the query, both LSTM and LSTM-Jump know what the question is. However, LSTM still has to process the remaining 20 sentences and thus at the very end of the last sentence, the long dependency between the question and output might become weak that the prediction is hampered. On the contrary, the question can guide LSTM-Jump on how to read selectively and stop early when the answer is clear. Therefore, when it comes to the output stage, the "memory" is both fresh and uncluttered that a more accurate answer is likely to be picked.

In the following, we show two examples of how the model reads the context given a query (bold face sentences are those read by our model in the increasing order). XXXXX is the missing word we want to fill. Note that due to truncation, a few sentences might look uncompleted.

Example 1 In the first example, the exact answer appears in the context multiple times, which makes the task relatively easy, as long as the reader has captured their occurrences.

(a) *Query:* 'XXXXX!

(b) *Context:*

1. **said Big Klaus, and he ran off at once to Little Klaus.**
2. 'Where did you get so much money from?'
3. **'Oh, that was from my horse-skin.**
4. I sold it yesterday evening.'
5. 'That 's certainly a good price!'

6. said Big Klaus; and running home in great haste, he took an axe, knocked all his four
7. **'Skins!**
8. **skins!**
9. Who will buy skins?'
10. he cried through the streets.
11. All the shoemakers and tanners came running to ask him what he wanted for them.'
12. **A bushel of money for each,' said Big Klaus.**
13. 'Are you mad?'
14. **they all exclaimed.**
15. 'Do you think we have money by the bushel?'
16. 'Skins!
17. skins!
18. Who will buy skins?'
19. he cried again, and to all who asked him what they cost, he answered,' A bushel
20. 'He is making game of us,' they said; and the shoemakers seized their yard measures and

(c) *Candidates:* Klaus | Skins | game | haste | head | home | horses | money | price| streets

(d) *Answer:* Skins

The reading behavior might be interpreted as follows. The model tries to search for clues, and after reading sentence 8, it realizes that the most plausible answer is "Klaus" or "Skins", as they both appear twice. "Skins" is more likely to be the answer as it is followed by a "!". The model searches further to see if "Klaus!" is mentioned somewhere, but it only finds "Klaus" without "!" for the third time. After the last attempt at sentence 14, it is confident about the answer and stops to output with "Skins".

Example 2 In this example, the answer is illustrated by a word "nuisance" that does not show up in the context at all. Hence, to answer the query, the model has to understand the meaning of both the query and context and locate the synonym of "nuisance", which is not merely verbatim and thus much harder than the previous example. Nevertheless, our model is still able to make a right choice while reading much fewer sentences.

(a) *Query:* Yes, I call XXXXX a nuisance.

(b) *Context:*

1. **But to you and me it would have looked just as it did to Cousin Myra – a very discontented**
2. "I'm awfully glad to see you, Cousin Myra, "explained Frank carefully, "and your
3. **But Christmas is just a bore – a regular bore."**
4. **That was what Uncle Edgar called things that didn't interest him, so that Frank felt pretty sure of**
5. **Nevertheless, he wondered uncomfortably what made Cousin Myra smile so queerly.**

6. **“Why, how dreadful!”**
7. she said brightly.
8. “I thought all boys and girls looked upon Christmas as the very best time in the year.”
9. “We don’t, ”said Frank gloomily.
10. **“It’s just the same old thing year in and year out.**
11. We know just exactly what is going to happen.
12. We even know pretty well what presents we are going to get.
13. And Christmas Day itself is always the same.
14. We’ll get up in the morning , and our stockings will be full of things, and half of
15. Then there ’s dinner.
16. It ’s always so poky.
17. And all the uncles and aunts come to dinner – just the same old crowd, every year, and
18. Aunt Desda always says, ‘Why, Frankie, how you have grown!’
19. She knows I hate to be called Frankie.
20. And after dinner they’ll sit round and talk the rest of the day, and that’s all.

(c) *Candidates:* Christmas | boys | day | dinner | half | interest | rest | stockings | things | uncles

(d) *Answer:* Christmas

The reading behavior can be interpreted as follows. After reading the query, our model realizes that the answer should be something like a nuisance. Then it starts to process the text. Once it hits sentence 3, it may begin to consider “Christmas” as the answer, since “bore” is a synonym of “nuisance”. Yet the model is not 100% sure, so it continues to read, very conservatively – it does not jump for the next three sentences. After that, the model gains more confidence on the answer “Christmas” and it makes a large jump to see if there is something that can turn over the current hypothesis. It turns out that the last-read sentence is still talking about Christmas with a negative voice. Therefore, the model stops to take “Christmas” as the output.

2.5 Discussion

In this chapter, we focus on learning how to skim text for fast reading. In particular, we propose a “jumping” model that after reading every few tokens, it decides how many tokens should be skipped by sampling from a softmax. Such jumping behavior is modeled as a discrete decision making process, which can be trained by reinforcement learning algorithm such as REINFORCE. In four different tasks with six datasets (one synthetic and five real), we test the efficiency of the proposed method on various levels of text jumping, from character to word and then to sentence. The results indicate our model is several times faster than, while the accuracy is on par with the baseline LSTM model.

Chapter 3

Recurrency-Free Model for Fully Parallel Computation

3.1 Introduction

There has been a surge of recent interest in the tasks of machine reading comprehension and automated question answering. Over the past few years, significant progress has been made with end-to-end models showing promising results on many challenging datasets. The most successful models generally employ two key ingredients: (1) a recurrent model to process sequential inputs, and (2) an attention component to cope with long term interactions. A successful combination of these two ingredients is the Bidirectional Attention Flow (BiDAF) model by [164], which achieves strong results on the SQuAD dataset [147]. A weakness of these models is that they are often slow for both training and inference due to their recurrent nature, especially when applied to long sequences. The expensive training not only leads to high turnaround time for experimentation and limits researchers from rapid iteration, but also prevents the models from being trained on larger datasets. Meanwhile the slow inference prevents the machine comprehension systems from being deployed in real-time applications.

In this chapter, aiming to make the machine comprehension fast, we propose to remove the recurrent nature of these models. We instead exclusively use convolutions and self-attentions as the building blocks of encoders that separately encodes the query and the context. Then we learn the interactions between the context and the question by standard attentions [11, 164, 203]. The resulting representation is encoded again with our recurrence-free encoder before finally decoding to the probability of each position being the start or end of the answer span. We call the proposed architecture QANet, which is depicted in Figure 3.1.

The key motivation behind the design of our model is the following: convolution captures the local structure of the text, while the self-attention learns the global interaction between each pair of words. The additional context-query attention is a standard module to construct the query-aware context vector for each position in the context paragraph, which is used in the subsequent modeling layers. The feed-forward nature of our architecture speeds up the model significantly. In our experiments on the SQuAD dataset, our model is 3x to 13x faster in training and 4x to 9x faster in inference. As a simple comparison, our model can achieve the same accuracy (F1 score

of 77.0) as BiDAF model [164] within 3 hours of training, which otherwise would take about 15 hours. The speed-up gain also allows us to train the model with more iterations to achieve better results than competitive models. For instance, if we allow our model to train for 18 hours, it achieves an F1 score of 82.7 on the dev set, which is much better than [164], and is on par with best published results.

Given that our model trains fast, we can train it with much more data than other models. To further improve the model, we propose a data augmentation technique to automatically generate more training data. This technique paraphrases the examples by translating the original sentences from English to another language and then back to English, which not only enhances the number of training instances but also improves the diversity of the phrases.

On the SQuAD dataset, QANet trained with data augmentation achieves an F1 score of 84.6 on the test set, which is significantly better than the best published result of 81.8 by [71]. We also conduct ablation test to assess the effectiveness of each component of the model. In summary, the contributions of the paper are as follows:

- We propose an efficient reading comprehension model that is exclusively built up of convolutions and self-attentions. This combination results in a good accuracy, while achieving 3x to 13x speedup in training time as opposed to the RNN counterparts. The speedup gain makes our model the most promising candidate for scaling up to larger datasets.
- We propose a novel data augmentation technique to enrich the training data via paraphrasing. When trained with data augmentation, our model achieves the state-of-the-art accuracy on question answering on SQuAD and TriviaQA datasets.

3.2 Related Work

Machine reading comprehension and automated question answering has become an important topic in the NLP domain. Their popularity can be attributed to an increase in publicly available annotated datasets, such as SQuAD [147], TriviaQA [79], CNN/Daily News [63], WikiReading [64], Children Book Test [65], etc. A great number of end-to-end neural network models have been proposed to tackle these challenges, including BiDAF [164], r-net [188], DCN [203], ReasoNet [172], Document Reader [23], Interactive AoA Reader [33] and Reinforced Mnemonic Reader [71].

Recurrent Neural Networks (RNNs) have featured predominatnly in Natural Language Processing in the past few years. The sequential nature of the text coincides with the design philosophy of RNNs, and hence their popularity. In fact, all the reading comprehension models mentioned above are based on RNNs. Despite being common, the sequential nature of RNN prevent parallel computation, as tokens must be fed into the RNN in order. Another drawback of RNNs is difficulty modeling long dependencies, although this is somewhat alleviated by the use of Gated Recurrent Unit [27] or Long Short Term Memory architectures [69]. For simple tasks such as text classification, with reinforcement learning techniques, models [210] have been proposed to skip irrelevant tokens to both further address the long dependencies issue and speed up the procedure. However, it is not clear if such methods can handle complicated tasks such as Q&A. The reading comprehension task considered in this paper always needs to deal with long text, as the context paragraphs may be hundreds of words long. Recently, attempts have

been made to replace the recurrent networks by full convolution or full attention architectures [52, 82, 171, 183]. Those models have been shown to be not only faster than the RNN architectures, but also effective in other tasks, such as text classification, machine translation or sentiment analysis.

To the best of our knowledge, our paper is the first work to achieve both *fast* and *accurate* reading comprehension model, by discarding the recurrent networks in favor of feed forward architectures. Our paper is also the first to mix self-attention and convolutions, which proves to be empirically effective and achieves a significant gain of 2.7 F1. Note that [146] recently proposed to accelerate reading comprehension by avoiding bi-directional attention and making computation conditional on the search beams. Nevertheless, their model is still based on the RNNs and the accuracy is not competitive, with an EM 68.4 and F1 76.2. [193] also tried to build a fast Q&A model by deleting the context-query attention module. However, it again relied on RNN and is thus intrinsically slower than ours. The elimination of attention further has sacrificed the performance (with EM 68.4 and F1 77.1).

Data augmentation has also been explored in natural language processing. For example, [218] proposed to enhance the dataset by replacing the words with their synonyms and showed its effectiveness in text classification. [146] suggested using type swap to augment the SQuAD dataset, which essentially replaces the words in the original paragraph with others with the same type. While it was shown to improve the accuracy, the augmented data has the same syntactic structure as the original data, so they are not sufficiently diverse. [223] improved the diversity of the SQuAD data by generating more questions. However, as reported by [188], their method did not help improve the performance. The data augmentation technique proposed in this paper is based on paraphrasing the sentences by translating the original text back and forth. The major benefit is that it can bring more syntactical diversity to the enhanced data.

3.3 The Model

In this section, we first formulate the reading comprehension problem and then describe the proposed model QANet: it is a feedforward model that consists of only convolutions and self-attention, a combination that is empirically effective, and is also a novel contribution of our work.

3.3.1 Problem Formulation

The reading comprehension task considered in this paper, is defined as follows. Given a context paragraph with n words $C = \{c_1, c_2, ..., c_n\}$ and the query sentence with m words $Q = \{q_1, q_2, ..., q_m\}$, output a span $S = \{c_i, c_{i+1}, ..., c_{i+j}\}$ from the original paragraph C. In the following, we will use x to denote both the original word and its embedded vector, for any $x \in C, Q$.

3.3.2 Model Overview

The high level structure of our model is similar to most existing models that contain five major components: an embedding layer, an embedding encoder layer, a context-query attention layer,

a model encoder layer and an output layer, as shown in Figure 3.1. These are the standard building blocks for most, if not all, existing reading comprehension models. However, the major differences between our approach and other methods are as follow: For both the embedding and modeling encoders, we only use convolutional and self-attention mechanism, discarding RNNs, which are used by most of the existing reading comprehension models. As a result, our model is much faster, as it can process the input tokens in parallel. Note that even though self-attention has already been used extensively in [183], the combination of convolutions and self-attention is novel, and is significantly better than self-attention alone and gives 2.7 F1 gain in our experiments. The use of convolutions also allows us to take advantage of common regularization methods in ConvNets such as stochastic depth (layer dropout) [72], which gives an additional gain of 0.2 F1 in our experiments.

In detail, our model consists of the following five layers:

1. Input Embedding Layer. We adopt the standard techniques to obtain the embedding of each word w by concatenating its word embedding and character embedding. The word embedding is fixed during training and initialized from the $p_1 = 300$ dimensional pre-trained GloVe [141] word vectors, which are fixed during training. All the out-of-vocabulary words are mapped to an <UNK> token, whose embedding is trainable with random initialization. The character embedding is obtained as follows: Each character is represented as a trainable vector of dimension $p_2 = 200$, meaning each word can be viewed as the concatenation of the embedding vectors for each of its characters. The length of each word is either truncated or padded to 16. We take maximum value of each row of this matrix to get a fixed-size vector representation of each word. Finally, the output of a given word x from this layer is the concatenation $[x_w; x_c] \in \mathbf{R}^{p_1+p_2}$, where x_w and x_c are the word embedding and the convolution output of character embedding of x respectively. Following [164], we also adopt a two-layer highway network [180] on top of this representation. For simplicity, we also use x to denote the output of this layer.

2. Embedding Encoder Layer. The encoder layer is a stack of the following basic building block: [convolution-layer $\times$ # + self-attention-layer + feed-forward-layer], as illustrated in the upper right of Figure 3.1. We use depthwise separable convolutions [26] [80] rather than traditional ones, as we observe that it is memory efficient and has better generalization. The kernel size is 7, the number of filters is $d = 128$ and the number of conv layers within a block is 4. For the self-attention-layer, we adopt the multi-head attention mechanism defined in [183] which, for each position in the input, called the query, computes a weighted sum of all positions, or keys, in the input based on the similarity between the query and key as measured by the dot product. The number of heads is 8 throughout all the layers. Each of these basic operations (conv/self-attention/ffn) is placed inside a *residual block*, shown lower-right in Figure 3.1. For an input x and a given operation f, the output is $f(layernorm(x)) + x$, meaning there is a full identity path from the input to output of each block, where layernorm indicates layer-normalization proposed in [10]. The total number of encoder blocks is 1. Note that the input of this layer is a vector of dimension $p_1 + p_2 = 500$ for each individual word, which is immediately mapped to $d = 128$ by a one-dimensional convolution. The output of this layer is a also of dimension $d = 128$.

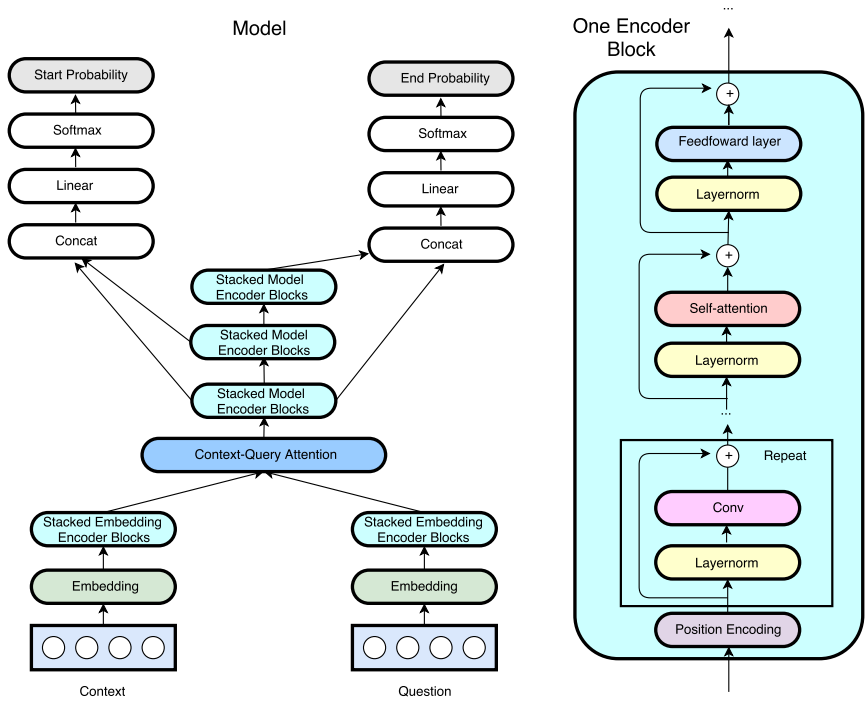

Figure 3.1: An overview of the QANet architecture (left) which has several Encoder Blocks. We use the same Encoder Block (right) throughout the model, only varying the number of convolutional layers for each block. We use layernorm and residual connection between every layer in the Encoder Block. We also share weights of the context and question encoder, and of the three output encoders. A positional encoding is added to the input at the beginning of each encoder layer consisting of sin and cos functions at varying wavelengths, as defined in [183]. Each sub-layer after the positional encoding (one of convolution, self-attention, or feed-forward-net) inside the encoder structure is wrapped inside a residual block.

3. Context-Query Attention Layer. This module is standard in almost every previous reading comprehension models such as [193] and [23]. We use C and Q to denote the encoded context and query. The context-to-query attention is constructed as follows: We first computer the similarities between each pair of context and query words, rendering a similarity matrix $S \in \mathbf{R}^{n \times m}$. We then normalize each row of S by applying the softmax function, getting a matrix $\overline{S}$. Then the context-to-query attention is computed as $A = \overline{S} \cdot Q^T \in \mathbf{R}^{n \times d}$. The similarity function used here is the trilinear function [164]:

$$f(q, c) = W_0[q, c, q \odot c],$$

where $\odot$ is the element-wise multiplication and W_0 is a trainable variable.

Most high performing models additionally use some form of query-to-context attention, such as BiDaF [164] and DCN [203]. Empirically, we find that, the DCN attention can provide a little benefit over simply applying context-to-query attention, so we adopt this strategy. More concretely, we compute the column normalized matrix $\overline{\overline{S}}$ of S by softmax function, and the query-to-context attention is $B = \overline{S} \cdot \overline{\overline{S}}^T \cdot C^T$.

4. Model Encoder Layer. Similar to [164], the input of this layer at each position is $[c, a, c \odot a, c \odot b]$, where a and b are respectively a row of attention matrix A and B. The layer parameters are the same as the Embedding Encoder Layer except that convolution layer number is 2 within a block and the total number of blocks are 10. We share weights between each of the 3 repetitions of the model encoder. So the total Model Encoder Layer has $10 \times (1 + 1 + 2) \times 3 = 120$ layers.

5. Output layer. This layer is task-specific. Each example in SQuAD is labeled with a span in the context containing the answer. We adopt the strategy of [164] to predict the probability of each position in the context being the start or end of an answer span. More specifically, the probabilities of the starting and ending position are modeled as

$$p^1 = softmax(W_1[M_0; M_1]), \;\; p^2 = softmax(W_2[M_0; M_2]),$$

where W_1 and W_2 are two trainable variables and M_0, M_1, M_2 are respectively the outputs of the three model encoders, from bottom to top. The score of a span is the product of its start position and end position probabilities. Finally, the objective function is defined as the negative sum of the log probabilities of the predicted distributions indexed by true start and end indices, averaged over all the training examples:

$$L(\theta) = -\frac{1}{N} \sum_{i}^{N} \left[\log(p^1_{y^1_i}) + \log(p^2_{y^2_i}) \right],$$

where y^1_i and y^2_i are respectively the groundtruth starting and ending position of example i, and θ contains all the trainable variables. The proposed model can be customized to other comprehension tasks, e.g. selecting from the candidate answers, by changing the output layers accordingly.

Combining all the five components, our model has over 130 layers, which is by far the deepest neural network for NLP.

Inference. At inference time, the predicted span (s, e) is chosen such that $p^1_s p^2_e$ is maximized and $s \leq e$. Standard dynamic programming can obtain the result with linear time.

3.4 Data Augmentation by Backtranslation

Since our QAnet model trains fast, we consider training the model using additional automatically generated data. We enrich the training dataset by using a simple data augmentation technique described here. The key idea is to use two machine translation models, one from English to *French* (or any other pivot language) and another from *French* back to English, to obtain paraphrases of the text inputs. This general approach helps automatically increasing the amount of training data for any natural language task including the reading comprehension task considered here. With more training data, we expect to enable better regularization of the models. The data augmentation process is illustrated in Figure 3.2 with French as the pivot language.

In this work, we make use of the attention-based neural machine translation (NMT) models [11, 107], which have demonstrated excellent translation quality [198], as the core components of the data augmentation pipeline. Specifically, we utilize the publicly available code[1]

[1] `https://github.com/tensorflow/nmt`

provided by Luong et al. [108], which replicates the Google's NMT (GNMT) systems [198]. We train 4-layer GNMT models on the public WMT data for both English-French[2] (36M sentence pairs) and English-German[3] (4.5M sentence pairs). All data have been tokenized and split into subword units as described in [108]. All models share the same hyperparameters[4] and are trained with different numbers of steps, 2M for English-French and 340K for English-German. Our English-French systems achieve 36.7 BLEU on newstest2014 for translation to French and 35.9 BLEU for the reverse direction. Our English-German model achieves 27.6 BLEU on newstest2014 for translation to German and 29.9 BLEU for the reverse direction.

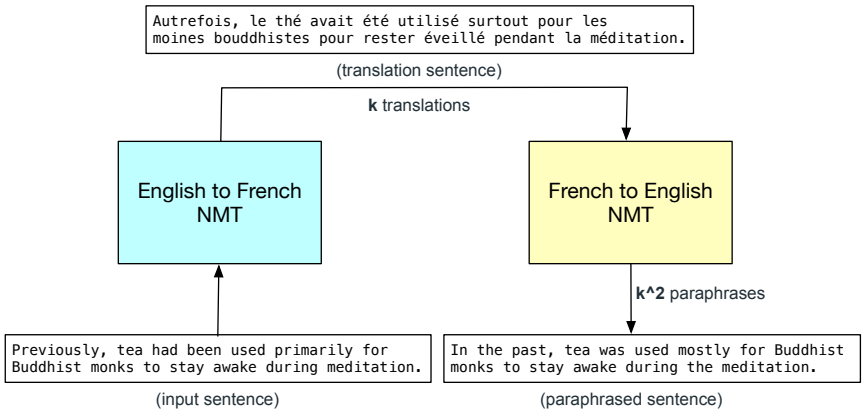

Figure 3.2: An illustration of the data augmentation process with French as the pivot language. Let k denote the beam width, which represents the number of translations generated by the NMT system in each phase. In total for a given English input, we will generate k^2 paraphrases.

Assuming that French is our pivot language, our paraphrase process works as follows. First, we feed an input sequence into the beam decoder of an English-to-French model to obtain k French translations. Each of the French translations is then passed through the beam decoder of a reversed translation model to obtain a total of k^2 paraphrases of the input sequence.

While the concept of backtranslation has been introduced before, it is often used to improve machine translation Sennrich et al. [163] or to enable intrinsic paraphrase evaluations Mallinson et al. [114], Wieting et al. [195]. Our approach is a novel application of backtranslation to enrich the training data for down-stream NLP tasks, in this case, question answering (Q&A). The independent work of [44] also uses paraphrasing techniques to improve Q&A, but, they only paraphrase questions and did not consider the data augmentation aspects that we leverage in this paper.

Handling SQuAD Documents and Answers. We now discuss our specific procedure for the SQuAD dataset, which is essential for the best performance gains. Remember that each training example of SQuAD is a triple of (d, q, a) in which document d is a multi-sentence paragraph that

[2] http://www.statmt.org/wmt14/

[3] http://www.statmt.org/wmt16/

[4] https://github.com/tensorflow/nmt/blob/master/nmt/standard_hparams/wmt16_gnmt_4_layer.json

includes the answer a. When paraphrasing, we keep the question q unchanged (to avoid accidentally changing its meaning) and generate new triples of (d', q, a') such that the new document d' has the new answer a' in it. The procedure happens in two steps: (i) *document paraphrasing* – paraphrase d into d' and (b) *answer extraction* – extract a' from d' that closely matches a.

For document paraphrasing, we first split the paragraphs into sentences and paraphrase individual sentences independently. We use $k = 5$, so each sentence has 25 paraphrases. A new document d' is formed by simply replacing each sentence in d with a randomly selected paraphrase. An obvious issue with this simple approach is that the original answer a might no longer appear in d'.

Our answer extraction process addresses the aforementioned issue. Let s be the original sentence that contains the original answer a and s' be its paraphrase. We identify the newly-paraphrased answer with simple heuristics as follows. Character-level 2-gram scores are computed between each word in s' and the start / end words of a to find start and end positions of possible answers in s'. Among all candidate paraphrased answer, the one with the highest character 2-gram score with respect to a is selected as the new answer a'. Table 3.1 shows an example of the new answer found by this process.[5]

	Sentence that contains an answer	Answer
Original	All of the departments in the College of Science offer PhD programs, except for the Department of Pre-Professional Studies.	Department of Pre-Professional Studies
Paraphrase	All departments in the College of Science offer PHD programs with the exception of the Department of Preparatory Studies.	Department of Preparatory Studies

Table 3.1: Comparison between answers in original sentence and paraphrased sentence.

The quality and diversity of paraphrases impact the effectiveness of the data augmentation method. We believe it is still possible to improve the quality and diversity of our method. One can improve the quality by using better translation models. For example, we find that paraphrases much longer than our models' maximum sequence length during training to have coverage issues. The diversity can be improved by sampling from translation models rather than beam search and paraphrasing questions and answers in the dataset as well. In addition, we can combine this method with other data augmentation methods, such as the type swap method [146], to acquire more diversity in paraphrases.

In our experiments, we observe that the proposed data augmentation can result in a significant accuracy improvement. We believe that this technique is applicable to other supervised natural language processing tasks, especially when the training dataset is small or the model is big.

[5]We also define a minimum threshold for elimination. If there is no answer with 2-gram score higher than the threshold, we remove the paraphrase s' from our sampling process. If all paraphrases of a sentence are eliminated, no sampling will be performed for that sentence.

3.5 Experiments

In this section, we conduct experiments to study the performance of our model and the data augmentation technique. We will primarily benchmark our model on the SQuAD dataset [147], considered to be one of the most competitive datasets in Q&A. We also conduct similar studies on TriviaQA [79], another Q&A dataset, to show that the effectiveness and efficiency of our model are general.

3.5.1 Experiments on SQuAD

Dataset and Experimental Settings

Dataset. We consider the Stanford Question Answering Dataset (SQuAD) [147] for machine reading comprehension.[6] SQuAD contains 107.7K query-answer pairs, with 87.5K for training, 10.1K for validation, and another 10.1K for testing. The typical length of the paragraphs is around 250 while the question is of 10 tokens although there are exceptionally long cases. Only the training and validation data are publicly available, while the test data is hidden that one has to submit the code to a Codalab and work with the authors of [147] to retrieve the final test score. In our experiments, we report the test set result of our best single model.[7] For further analysis, we only report the performance on the validation set, as we do not want to probe the unseen test set by frequent submissions. According to the observations from our experiments and previous works, such as [23, 164, 188, 203], the validation score is well correlated with the test score.

Data Preprocessing. We use the NLTK tokenizer to preprocess the data.[8] The maximum context length is set to 400 and any paragraph longer than that would be discarded. During training, we batch the examples by length and dynamically pad the short sentences with special symbol `<PAD>`. The maximum answer length is set to 30. We use the pretrained 300-D word vectors GLoVe [141], and all the out-of-vocabulary words are replace with `<UNK>`, whose embedding is updated during training. Each character embedding is randomly initialized as a 200-D vector, which is updated in training as well. We generate two additional augmented datasets obtained from Section 3.4, which contain 140K and 240K examples and are denoted as "data augmentation $\times$ 2" and "data augmentation $\times$ 3" respectively, including the original data.

Training details. We employ two types of standard regularizations. First, we use L2 weight decay on all the trainable variables, with parameter $\lambda = 3 \times 10^{-7}$. We additionally use dropout on word, character embeddings and between layers, where the word and character dropout rates are 0.1 and 0.05 respectively, and the dropout rate between every two layers is 0.1. We also adopt the stochastic depth method (layer dropout) [72] within each embedding or model encoder

[6] SQuAD leaderboard: `https://rajpurkar.github.io/SQuAD-explorer/`

[7] On the leaderboard of SQuAD, there are many strong candidates in the "ensemble" category with high EM/F1 scores. Although it is possible to improve the results of our model using ensembles, we focus on the "single model" category and compare against other models with the same category.

[8] NLTK implementation: `http://www.nltk.org/`

layer, where sublayer l has survival probability $p_l = 1 - \frac{l}{L}(1 - p_L)$ where L is the last layer and $p_L = 0.9$.

The hidden size and the convolution filter number are all 128, the batch size is 32, training steps are 150K for original data, 250K for "data augmentation $\times$ 2", and 340K for "data augmentation $\times$ 3". The numbers of convolution layers in the embedding and modeling encoder are 4 and 2, kernel sizes are 7 and 5, and the block numbers for the encoders are 1 and 7, respectively.

We use the ADAM optimizer [83] with $\beta_1 = 0.8, \beta_2 = 0.999, \epsilon = 10^{-7}$. We use a learning rate warm-up scheme with an inverse exponential increase from 0.0 to 0.001 in the first 1000 steps, and then maintain a constant learning rate for the remainder of training. Exponential moving average is applied on all trainable variables with a decay rate 0.9999.

Finally, we implement our model in Python using Tensorflow [1] and carry out our experiments on an NVIDIA p100 GPU.[9]

Results

Accuracy. The F1 and Exact Match (EM) are two evaluation metrics of accuracy for the model performance. F1 measures the portion of overlap tokens between the predicted answer and groundtruth, while exact match score is 1 if the prediction is exactly the same as groundtruth or 0 otherwise. We show the results in comparison with other methods in Table 3.2. To make a fair and thorough comparison, we both report both the published results in their latest papers/preprints and the updated but not documented results on the leaderboard. We deem the latter as the unpublished results. As can be seen from the table, the accuracy (EM/F1) performance of our model is on par with the state-of-the-art models. In particular, our model trained on the original dataset outperforms all the documented results in the literature, in terms of both EM and F1 scores (see second column of Table 3.2). When trained with the augmented data with proper sampling scheme, our model can get significant gain 1.5/1.1 on EM/F1. Finally, our result on the official test set is 76.2/84.6, which significantly outperforms the best documented result 73.2/81.8. If we build our model on top of the contextual embedding ELMo [142] and CoVe [117], we can get an even higher accuracy 82.5/89.3. Finally, the ensemble of 14 QANet with different initializations and numbers of layers can achieve 84.5/90.5 on EM/F1, which is the best performance on the leaderboard from March to September 2018 until BERT model [43] came out. Notably, QANet is among the first batch of models that exceed the human performance on EM (82.3).

Speedup over RNNs. To measure the speedup of our model against the RNN models, we also test the corresponding model architecture with each encoder block replaced with a stack of bidirectional LSTMs as is used in most existing models. Specifically, each (embedding and model) encoder block is replaced with a 1, 2, or 3 layer Bidirectional LSTMs respectively, as such layer numbers fall into the usual range of the reading comprehension models [23]. All of these LSTMs have hidden size 128. The results of the speedup comparison are shown in

[9]TensorFlow implementation: `https://www.tensorflow.org/`

[11]The scores are collected from the latest version of the documented related work on Oct 27, 2017.

[12]The scores are collected from the leaderboard on Oct 27, 2017.

Table 3.3. We can easily see that our model is significantly faster than all the RNN based models and the speedups range from 3 to 13 times in training and 4 to 9 times in inference.

Speedup over BiDAF model. In addition, we also use the same hardware (a NVIDIA p100 GPU) and compare the training time of getting the same performance between our model and the BiDAF model[13][164], a classic RNN-based model on SQuAD. We mostly adopt the default settings in the original code to get its best performance, where the batch sizes for training and inference are both 60. The only part we changed is the optimizer, where Adam with learning 0.001 is used here, as with Adadelta we got a bit worse performance. The result is shown in Table 3.4 which shows that our model is 4.3 and 7.0 times faster than BiDAF in training and inference speed. Besides, we only need one fifth of the training time to achieve BiDAF's best F1 score (77.0) on dev set.

Ablation Study and Analysis

We conduct ablation studies on components of the proposed model, and investigate the effect of augmented data. The validation scores on the development set are shown in Table 3.5. As can be seen from the table, the use of convolutions in the encoders is crucial: both F1 and EM drop drastically by almost 3 percent if it is removed. Self-attention in the encoders is also a necessary component that contributes 1.4/1.3 gain of EM/F1 to the ultimate performance. We interpret these phenomena as follows: the convolutions capture the local structure of the context while the self-attention is able to model the global interactions between text. Hence they are complimentary to but cannot replace each other. The use of separable convolutions in lieu of tradition convolutions also has a prominent contribution to the performance, which can be seen by the slightly worse accuracy caused by replacing separable convolution with normal convolution.

The Effect of Data Augmentation. We additionally perform experiments to understand the values of augmented data as their amount increases. As the last block of rows in the table shows, data augmentation proves to be helpful in further boosting performance. Making the training data twice as large by adding the En-Fr-En data only (ratio 1:1 between original training data and augmented data, as indicated by row "data augmentation $\times$ 2 (1:1:0)") yields an increase in the F1 by 0.5 percent. While adding more augmented data with French as a pivot does not provide performance gain, injecting additional augmented data En-De-En of the same amount brings another 0.2 improvement in F1, as indicated in entry "data augmentation $\times$ 3 (1:1:1)". We may attribute this gain to the diversity of the new data, which is produced by the translator of the new language.

The Effect of Sampling Scheme. Although injecting more data beyond $\times$ 3 does not benefit the model, we observe that a good sampling ratio between the original and augmented data during training can further boost the model performance. In particular, when we increase the sampling weight of augmented data from (1:1:1) to (1:2:1), the EM/F1 performance drops by 0.5/0.3. We conjecture that it is due to the fact that augmented data is noisy because of the back-translation,

[13]The code is directly downloaded from `https://github.com/allenai/bi-att-flow`

so it should not be the dominant data of training. We confirm this point by increasing the ratio of the original data from (1:2:1) to (2:2:1), where 0.6/0.5 performance gain on EM/F1 is obtained. Then we fix the portion of the augmented data, and search the sample weight of the original data. Empirically, the ratio (3:1:1) yields the best performance, with 1.5/1.1 gain over the base model on EM/F1. This is also the model we submitted for test set evaluation.

Error Analysis. Although our ensemble model has achieved the so-called human performance on the EM metric, we are still interested in investigating what kind of wrong predictions QANet makes to further improve the model. Here we pick out a representative case in Table 3.6. In this example, the question is "What year marked the lowest ratings for the drama Lost?". We train 14 QANet variants with different initializations and numbers of layers, all of which make the same wrong prediction. We can see that QANet is able to find a number corresponding to a year, but this number is wrong. Actually, QANet seems to first locate the term "lowest ratings" appearing in both the question and the context, and then pick the number closest to the term in the context as the predicted answer. That probably tells the fact that QANet is essentially doing simple pattern matching and it might not truly "understand" the question. Therefore, it can be easily misled.

To address those issues, we might need to either inject the hand-crafted co-reference features or impose multi-task objectives on the pretraining such as ELMo and BERT. However, such modifications might also introduce bias to the model, resulting in other issues. Hence, it could be an interesting future work to examine the trade-offs.

Robustness Study

In the following, we conduct experiments on the adversarial SQuAD dataset [77] to study the robustness of the proposed model. In this dataset, one or more sentences are appended to the original SQuAD context of test set, to intentionally mislead the trained models to produce wrong answers. However, the model is agnostic to those adversarial examples during training. We focus on two types of misleading sentences, namely, AddSent and AddOneSent. AddSent generates sentences that are similar to the question, but not contradictory to the correct answer, while AddOneSent adds a random human-approved sentence that is not necessarily related to the context.

The model in use is exactly the one trained with the original SQuAD data (the one getting 84.6 F1 on test set), but now it is submitted to the adversarial server for evaluation. The results are shown in Table 3.7, where the F1 scores of other models are all extracted from [77].[14] Again, we only compare the performance of single models. From Table 3.7, we can see that our model is on par with the state-of-the-art model Mnemonic, while significantly better than other models by a large margin. The robustness of our model is probably because it is trained with augmented data. The injected noise in the training data might not only improve the generalization of the model but also make it robust to the adversarial sentences.

[14]Only F1 scores are reported in [77]

3.5.2 Experiments on TriviaQA

In this section, we test our model on another dataset TriviaQA [79], which consists of 650K context-query-answer triples. There are 95K distinct question-answer pairs, which are authored by Trivia enthusiasts, with 6 evidence documents (context) per question on average, which are either crawled from Wikipedia or Web search. Compared to SQuAD, TriviaQA is more challenging in that: 1) its examples have much longer context (2895 tokens per context on average) and may contain several paragraphs, 2) it is much noisier than SQuAD due to the lack of human labeling, 3) it is possible that the context is not related to the answer at all, as it is crawled by key words.

In this paper, we focus on testing our model on the subset consisting of answers from Wikipedia. According to the previous work [71, 79, 136], the same model would have similar performance on both Wikipedia and Web, but the latter is five time larger. To keep the training time manageable, we omit the experiment on Web data.

Due to the multi-paragraph nature of the context, researchers also find that simple hierarchical or multi-step reading tricks, such as first predicting which paragraph to read and then apply models like BiDAF to pinpoint the answer within that paragraph [29], can significantly boost the performance on TriviaQA. However, in this paper, we focus on comparing with the single-paragraph reading baselines only. We believe that our model can be plugged into other multi-paragraph reading methods to achieve the similar or better performance, but it is out of the scope of this paper.

The Wikipedia sub-dataset contains around 92K training and 11K development examples. The average context and question lengths are 495 and 15 respectively. In addition to the full development set, the authors of [79] also pick a verified subset that all the contexts inside can answer the associated questions. As the text could be long, we adopt the data processing similar to [71, 79]. In particular, for training and validation, we randomly select a window of length 256 and 400 encapsulating the answer respectively. All the remaining setting are the same as SQuAD experiment, except that the training steps are set to 120K.

Accuracy. The accuracy performance on the development set is shown in Table 3.8. Again, we can see that our model outperforms the baselines in terms of F1 and EM on Full development set, and is on par with the state-of-the-art on the Verified dev set.

Speedup over RNNs. In addition to accuracy, we also benchmark the speed of our model against the RNN counterparts. As Table 3.9 shows, not surprisingly, our model has 3 to 11 times speedup in training and 3 to 9 times acceleration in inference, similar to the finding in SQuAD.

3.6 Discussion

In this chapter, we propose a fast and accurate end-to-end model, QANet, for machine reading comprehension. Our core innovation is to completely remove the recurrent networks in the encoder. The resulting model is fully feedforward, composed entirely of separable convolutions, attention, linear layers, and layer normalization, which is suitable for parallel computation. The

resulting model is both fast and accurate: It surpasses the best published results on SQuAD dataset while up to 13/9 times faster than a competitive recurrent models for a training/inference iteration. Additionally, we find that we are able to achieve significant gains by utilizing data augmentation consisting of translating context and passage pairs to and from another language as a way of paraphrasing the questions and contexts.

	Published[10]	LeaderBoard[11]
Single Model	EM / F1	EM / F1
LR Baseline[a]	40.4 / 51.0	40.4 / 51.0
Dynamic Chunk Reader[b]	62.5 / 71.0	62.5 / 71.0
Match-LSTM with Ans-Ptr[c]	64.7 / 73.7	64.7 / 73.7
Multi-Perspective Matching[d]	65.5 / 75.1	70.4 / 78.8
Dynamic Coattention Networks[e]	66.2 / 75.9	66.2 / 75.9
FastQA[f]	68.4 / 77.1	68.4 / 77.1
BiDAF[g]	68.0 / 77.3	68.0 / 77.3
SEDT[h]	68.1 / 77.5	68.5 / 78.0
RaSoR[i]	70.8 / 78.7	69.6 / 77.7
FastQAExt[j]	70.8 / 78.9	70.8 / 78.9
ReasoNet[k]	69.1 / 78.9	70.6 / 79.4
Document Reader[l]	70.0 / 79.0	70.7 / 79.4
Ruminating Reader[m]	70.6 / 79.5	70.6 / 79.5
jNet[n]	70.6 / 79.8	70.6 / 79.8
Conductor-net	N/A	72.6 / 81.4
Interactive AoA Reader[o]	N/A	73.6 / 81.9
Reg-RaSoR	N/A	75.8 / 83.3
DCN+	N/A	74.9 / 82.8
AIR-FusionNet	N/A	76.0 / 83.9
R-Net[p]	72.3 / 80.7	76.5 /84.3
BiDAF + Self Attention + ELMo	N/A	77.9/ 85.3
Reinforced Mnemonic Reader[q]	73.2 / 81.8	73.2 / 81.8
Dev set: QANet	**73.6 / 82.7**	N/A
Dev set: QANet + data augmentation $\times 2$	**74.5 / 83.2**	N/A
Dev set: QANet + data augmentation $\times 3$	**75.1 / 83.8**	N/A
Test set: QANet + data augmentation $\times 3$	**76.2 / 84.6**	76.2 / 84.6
Test set: QANet + data augmentation $\times 3$ + ELMo + CoVe	**82.5 / 89.3**[12]	82.5 / 89.3
Test set: Ensemble of 14 QANets + ELMo + CoVe	**84.5 / 90.5**	84.5 / 90.5
Test set: Human Performance	82.3 / 91.2	82.3 / 91.2

Table 3.2: The performances of different models on SQuAD dataset. Models references: a.[147], b.[213], c.[187], d.[190] , e.[203] , f.[193], g.[164], h.[105], i.[95], j.[193], k.[172], l.[23], m.[53], n.[217], o.[33], p.[188], q.[71].

	QANet	RNN-1-128	Speedup	RNN-2-128	Speedup	RNN-3-128	Speedup
Training	**3.2**	1.1	**2.9x**	0.34	**9.4x**	0.24	**13.3x**
Inference	**8.1**	2.2	**3.7x**	1.3	**6.2x**	0.92	**8.8x**

Table 3.3: Speed comparison between our model and RNN-based models on SQuAD dataset, all with batch size 32. RNN-x-y indicates an RNN with x layers each containing y hidden units. Here, we use bidirectional LSTM as the RNN. The speed is measured by batches/second, so higher is faster.

	Train time to get 77.0 F1 on Dev set	Train speed	Inference speed
QANet	3 hours	102 samples/s	259 samples/s
BiDAF	15 hours	24 samples/s	37samples/s
Speedup	**5.0x**	**4.3x**	**7.0x**

Table 3.4: Speed comparison between our model and BiDAF [164] on SQuAD dataset.

	EM / F1	Difference to Base Model EM / F1
Base QANet	73.6 / 82.7	
- convolution in encoders	70.8 / 80.0	-2.8 / -2.7
- self-attention in encoders	72.2 / 81.4	-1.4 / -1.3
replace sep convolution with normal convolution	72.9 / 82.0	- 0.7 / -0.7
+ data augmentation $\times 2$ (1:1:0)	74.5 / 83.2	+0.9 / +0.5
+ data augmentation $\times 3$ (1:1:1)	74.8 / 83.4	+1.2 / +0.7
+ data augmentation $\times 3$ (1:2:1)	74.3 / 83.1	+0.7 / +0.4
+ data augmentation $\times 3$ (2:2:1)	74.9 / 83.6	+1.3 / +0.9
+ data augmentation $\times 3$ (2:1:1)	75.0 / 83.6	+1.4 / +0.9
+ data augmentation $\times 3$ (3:1:1)	**75.1 / 83.8**	**+1.5 / +1.1**
+ data augmentation $\times 3$ (4:1:1)	75.0 / 83.6	+1.4 / +0.9
+ data augmentation $\times 3$ (5:1:1)	74.9 / 83.5	+1.3 / +0.8

Table 3.5: An ablation study of data augmentation and other aspects of our model. The reported results are obtained on the *development set*. For rows containing entry "data augmentation", "$\times N$" means the data is enhanced to N times as large as the original size, while the ratio in the bracket indicates the sampling ratio among the original, English-French-English and English-German-English data during training.

Context	The network began running into some trouble in the ratings by **2010**. That year, the sixth and final season of Lost became the drama's **lowest-rated** season since its debut in **2004** ...
Question	What year marked the **lowest ratings** for the drama Lost?
Ground truth	2010
Prediction	2004

Table 3.6: Error analysis on SQuAD.

Single Model	AddSent	AddOneSent
Logistic[a]	23.2	30.4
Match[b]	27.3	39.0
SEDT[c]	33.9	44.8
DCR[d]	37.8	45.1
BiDAF[e]	34.3	45.7
jNet[f]	37.9	47.0
Ruminating[g]	37.4	47.7
RaSOR[h]	39.5	49.5
MPCM[i]	40.3	50.0
ReasoNet[j]	39.4	50.3
Mnemonic[k]	**46.6**	**56.0**
QANet	45.2	55.7

Table 3.7: The F1 scores on the adversarial SQuAD test set. Model references: a.[147], b.[187], c.[105], d.[213], e.[164], f.[217], g.[53], h.[95], i.[190], j.[172], k.[71].

	Full	Verified
Single Model	EM / F1	EM / F1
Random[a]	12.7 / 22.5	13.8 / 23.4
Classifier[b]	23.4 / 27.7	23.6 / 27.9
BiDAF[c]	40.3 / 45.7	46.5 /52.8
MEMEN[d]	43.2/ 46.9	49.3 / 55.8
M-Reader[e]	46.9/ 52.9*	54.5/ 59.5*
QANet	**51.1 / 56.6**	53.3/ 59.2

Table 3.8: The development set performances of different *single-paragraph* reading models on the Wikipedia domain of TriviaQA dataset. Note that * indicates the result on test set. Model references: a.[79], b.[79], c.[164], d.[136], e.[71].

	QANet	RNN-1-128	Speedup	RNN-2-128	Speedup	RNN-3-128	Speedup
Training	**1.8**	0.41	**4.4x**	0.20	**9.0x**	0.11	**16.4x**
Inference	**3.2**	0.89	**3.6x**	0.47	**6.8x**	0.26	**12.3x**

Table 3.9: Speed comparison between the proposed model and RNN-based models on TriviaQA Wikipedia dataset, all with batch size 32. RNN-x-y indicates an RNN with x layers each containing y hidden units. The RNNs used here are bidirectional LSTM. The processing speed is measured by batches/second, so higher is faster.

Part II

Efficient Algorithms for Large Scale Optimization

Chapter 4

Normalized Gradient Method for Training Deep Neural Networks

4.1 Introduction

Continuous optimization is a core technique for training non-convex sophisticated machine learning models such as deep neural networks [15]. Compared to convex optimization where a global optimal solution is expected, non-convex optimization usually aims to find a stationary point or a local optimal solution of an objective function by iterative algorithms. Among a large volume of optimization algorithms, first-order methods, which only iterate with the gradient information of objective functions, are widely used due to its relatively low requirement on memory space and computation time, compared to higher order algorithms. In many machine learning scenarios with large amount of data, the full gradient is still expensive to obtain, and hence the unbiased stochastic version will be adopted, as it is even more computationally efficient.

In this chapter, we are particularly interested in solving the deep neural network training problems with stochastic first order methods. Compared to other non-convex problems, deep neural network training additionally has the following challenge: gradient may be vanishing and/or exploding. More specifically, due to the chain rule (a.k.a. backpropagation), the original gradient in the low layers will become very small or very large because of the multiplicative effect of the gradients from the upper layers, which is usually all smaller or larger than 1. As the number of layers in the neural network increases, the phenomenon of vanishing or exploding gradients becomes more severe such that the iterative solution will converge slowly or diverge quickly.

We aim to alleviate this problem by block-wise stochastic *gradient normalization*, which is constructed via dividing the stochastic gradient by its norm. Here, each block essentially contains the variables of one layer in a neural network, so it can also be interpreted as layer-wise gradient normalization. Compared to the regular gradient, normalized gradient only provides an updating direction but does not incorporate the local steepness of the objective through its magnitude, which helps to control the change of the solution through a well-designed step length. Intuitively, as it constrains the magnitude of the gradient to be 1 (or a constant), it should to some extent prevent the gradient vanishing or exploding phenomenon. In fact, as showed in

[59, 97], normalized gradient descent (NGD) methods are more numerically stable and have better theoretical convergence properties than the regular gradient descent method in non-convex optimization.

Once the updating direction is determined, step size (learning rate) is the next important component in the design of first-order methods. While for convex problems there are some well studied strategies to find a stepsize ensuring convergence, for non-convex optimization, the choice of step size is more difficult and critical as it may either enlarge or reduce the impact of the aforementioned vanishing or exploding gradients.

Among different choices of step sizes, the constant or adaptive *feature-dependent* step sizes are widely adopted. On one hand, stochastic gradient descent (SGD) + momentum + constant step size has become the standard choice for training feed-forward networks such as Convolution Neural Networks (CNN). Ad-hoc strategies like decreasing the step size when the validation curve plateaus are well adopted to further improve the generalization quality. On the other hand, different from the standard step-size rule which multiplies a same number to each coordinate of gradient, the adaptive feature-dependent step-size rule multiplies different numbers to coordinates of gradient so that different parameters in the learning model can be updated in different paces. For example, the adaptive step size invented by [47] is constructed by aggregating each coordinates of historical gradients. As discussed by [47], this method can dynamically incorporate the frequency of features in the step size so that frequently occurring coordinates will have a small step sizes while infrequent features have long ones. The similar adaptive step size is proposed in [83] but the historical gradients are integrated into feature-dependent step size by a different weighting scheme.

In this chapter, we propose a generic framework using the mini-batch stochastic normalized gradient as the updating direction (like [59, 97]) and the step size is either constant or adaptive to each coordinate as in [47, 83]. Our framework starts with computing regular mini-batch stochastic gradient, which is immediately normalized layer-wisely. The normalized version is then plugged in the constant stepsize with occasional decay, such as SGD+momentum, or the adaptive step size methods, such as Adam [83] and AdaGrad [47]. The numerical results shows that normalized gradient always helps to improve the performance of the original methods especially when the network structure is deep. It seems to be the first thorough empirical study on various types of neural networks with this normalized gradient idea. Besides, although we focus our empirical studies on deep learning where the objective is highly non-convex, we also provide a convergence proof under this framework when the problem is convex and the stepsize is adaptive in the appendix. The convergence under the non-convex case will be a very interesting and important future work.

4.2 Related Work

A pioneering work on normalized gradient descent (NGD) method was by Nesterov [127] where it was shown that NGD can find a ϵ-optimal solution within $O(\frac{1}{\epsilon^2})$ iterations when the objective function is differentiable and quasi-convex. Kiwiel [84] and Hazan et al [59] extended NGD for upper semi-continuous (but not necessarily differentiable) quasi-convex objective functions and local-quasi-convex objective functions, respectively, and achieved the same iteration complexity.

Moreover, Hazan et al [59] showed that NGD's iteration complexity can be reduced to $O(\frac{1}{\epsilon})$ if the objective function is local-quasi-convex and locally-smooth. A stochastic NGD algorithm is also proposed by Hazan et al [59] which, if a mini-batch is used to construct the stochastic normalized gradient in each iteration, finds ϵ-optimal solution with a high probability for locally-quasi-convex functions within $O(\frac{1}{\epsilon^2})$ iterations. Levy [97] proposed a Saddle-Normalized Gradient Descent (Saddle-NGD) method, which adds a zero-mean Gaussian random noise to the stochastic normalized gradient periodically. When applied to strict-saddle functions with some additional assumption, it is shown [97] that Saddle-NGD can evade the saddle points and find a local minimum point approximately with a high probability.

Analogous yet orthogonal to the gradient normalization ideas have been proposed for the deep neural network training. For example, batch normalization [74] is used to address the internal covariate shift phenomenon in the during deep learning training. It benefits from making normalization a part of the model architecture and performing the normalization for each training mini-batch. Weight normalization [159], on the other hand, aims at a reparameterization of the weight vectors that decouples the length of those weight vectors from their direction. Recently [133] proposes to use path normalization, an approximate path-regularized steepest descent with respect to a path-wise regularizer related to max-norm regularization to achieve better convergence than vanilla SGD and AdaGrad. Perhaps the most related idea to ours is Gradient clipping. It is proposed in [139] to avoid the gradient explosion, by pulling the magnitude of a large gradient to a certain level. However, this method does not do anything when the magnitude of the gradient is small.

Adaptive step size has been studied for years in the optimization community. The most celebrated method is the line search scheme. However, while the exact line search is usually computational infeasible, the inexact line search also involves a lot of full gradient evaluation. Hence, they are not suitable for the deep learning setting. Recently, algorithms with adaptive step sizes start to be applied to the non-convex neural network training, such as AdaGrad [48], Adam [83] and RMSProp [66]. However they directly use the unnormalized gradient, which is different from our framework. [174] recently proposes to apply layer-wise specific step sizes, which differs from ours in that it essentially adds a term to the gradient rather than normalizing it. Recently [197] finds the methods with adaptive step size might converge to a solution with worse generalization. However, this is orthogonal to our focus in this paper.

4.3 Algorithm Framework

In this section, we present our algorithm framework to solve the following general problem:

$$\min_{x\in\mathbb{R}^d} f(x) = \mathbb{E}(F(x,\xi)), \tag{4.3.1}$$

where $x = (x^1, x^2, \ldots, x^B) \in \mathbb{R}^d$ with $x^i \in \mathbb{R}^{d_i}$ and $\sum_{i=1}^B d_i = d$, ξ is a random variable following some distribution $\mathbb{P}$, $F(\cdot,\xi)$ is a loss function for each ξ and the expectation $\mathbb{E}$ is taken over ξ. In the case where (4.3.1) models an empirical risk minimization problem, the distribution $\mathbb{P}$ can be the empirical distribution over training samples such that the objective function in (4.3.1) becomes a finite-sum function. Now our goal is to minimize the objective function f over

x, where x can be the parameters of a machine learning model when (4.3.1) corresponds to a training problem. Here, the parameters are partitioned into B blocks. The problem of training a neural network is an important instance of (4.3.1), where each block of parameters x^i can be viewed as the parameters associated to the ith layer in the network.

We propose the generic optimization framework in Algorithm 1. In iteration t, it firstly computes the partial (sub)gradient $F_i'(x_t, \xi_t)$ of F with respect to x^i for $i = 1, 2, \ldots,$ at $x = x_t$ with a mini-batch data ξ_t, and then normalizes it to get a partial direction $g_t^i = \frac{F_i'(x_t,\xi_t)}{\|F_i'(x_t,\xi_t)\|_2}$. We define $g_t = (g_t^1, g_t^2, \ldots, g_t^B)$. The next is to find d adaptive step sizes $\tau_t \in \mathbb{R}^d$ with each coordinate of τ_t corresponding to a coordinate of x. We also partition τ_t in the same way as x so that $\tau_t = (\tau_t^1, \tau_t^2, \ldots, \tau_t^B) \in \mathbb{R}^B$ with $\tau_t^i \in \mathbb{R}^{d_i}$. We use τ_t as step sizes to update x_t to x_{t+1} as $x_{t+1} = x_t - \tau_t \circ g_t$, where $\circ$ represents coordinate-wise (Hadamard) product. In fact, our framework can be customized to most of existing first order methods with fixed or adaptive step sizes, such as SGD, AdaGrad[47], RMSProp [66] and Adam[83], by adopting their step size rules respectively.

Algorithm 1 Generic Block-Normalized Gradient (BNG) Descent

1: Choose $x_1 \in \mathbb{R}^d$.
2: **for** $t = 1, 2, ...,$ **do**
3: Sample a mini-batch of data ξ_t and compute the partial stochastic gradient $g_t^i = \frac{F_i'(x_t,\xi_t)}{\|F_i'(x_t,\xi_t)\|_2}$
4: Let $g_t = (g_t^1, g_t^2, \ldots, g_t^B)$ and choose step sizes $\tau_t \in \mathbb{R}^d$.
5: $x_{t+1} = x_t - \tau_t \circ g_t$
6: **end for**

4.4 Convergence Analysis for A Concrete Case

As a concrete example for Algorithm 1, we present a simple modification of AdaGrad using block-wise normalized stochastic gradient in Algorithm 2, where $g_{1:t}$ is a matrix created by stacking g_1, g_2,... and g_t in columns and $g_{1:t,j} \in \mathbb{R}^t$ represents the jth row of $g_{1:t}$ for $j = 1, 2, \ldots, d$. Assuming the function F is convex over x for any ξ and following the analysis in [47], it is straightforward to show a $O(\frac{1}{\sqrt{T}})$ convergence rate of this modification. In the following, we denote $\|x\|_W := \sqrt{x^\top W x}$ as the the Mahalanobis norm associated to a $d \times d$ positive definite matrix W. The convergence property of Block-Normalized AdaGrad is presented in the following theorem.

Theorem 4.4.1 *Suppose F is convex over x, $\|F_i'(x_t, \xi_t)\|_2 \leq M_i$ and $\|x_t - x^*\|_\infty \leq D_\infty$ for all t for some constants M_i and $D_\infty > 0$ in Algorithm 2. Let $H_t = \delta I_d + diag(s_t)$ and $H_t^i =$*

Algorithm 2 AdaGrad with Block-Normalized Gradient (AdaGradBNG)

1: Choose $x_1 \in \mathbb{R}^d$, $\delta > 0$ and $\eta > 0$.
2: **for** $t = 1, 2, ...,$ **do**
3: Sample a mini-batch data ξ_t and compute the stochastic partial gradient $g_t^i = \frac{F_i'(x_t,\xi_t)}{\|F_i'(x_t,\xi_t)\|_2}$
4: Let $g_t = (g_t^1, g_t^2, \ldots, g_t^B)$, $g_{1:t} = [g_1, g_2, \ldots, g_t]$ and $s_t = (\|g_{1:t,1}\|_2, \|g_{1:t,2}\|_2, \ldots, \|g_{1:t,d}\|_2)$
5: Partition $s_t = (s_t^1, s_t^2, \ldots, s_t^B)$ in the same way as g_t.
6: Let $\tau_t = (\tau_t^1, \tau_t^2, \ldots, \tau_t^B)$ with[1] $\tau_t^i = \eta\|F_i'(x_t,\xi_t)\|_2(\delta \mathbf{1}_{d_i} + s_t^i)^{-1}$.
7: $x_{t+1} = x_t - \tau_t \circ g_t$
8: **end for**

$\delta I_{d_i} + diag(s_t^i)$ for $t = 1, 2, \ldots$ and $\bar{x}_T := \frac{1}{T}\sum_{t=1}^T x_t$. Algorithm 2 guarantees

$$\begin{aligned}\mathbb{E}[f(\bar{x}_T) - f(x^*)] &\leq \frac{\|x_1 - x^*\|_{H_1}^2}{2\eta T} + \frac{D_\infty^2\sqrt{Bd}}{2\eta\sqrt{T}} + \sum_{i=1}^B \frac{\eta\mathbb{E}\left[M_i^2 \sum_{j=d_1+d_2+\cdots+d_{i-1}+1}^{d_1+d_2+\cdots+d_i} \|g_{1:T,j}\|_2\right]}{T} \\ &\leq \frac{\|x_1 - x^*\|_{H_1}^2}{2\eta T} + \frac{D_\infty^2\sqrt{Bd}}{2\eta\sqrt{T}} + \sum_{i=1}^B \frac{\eta M_i^2\sqrt{d_i}}{\sqrt{T}}.\end{aligned}$$

Proof: It is easy to see that H_t is a positive definite and diagonal matrix. According to the updating scheme of x_{t+1} and the definitions of H_t, τ_t and g_t, we have

$$\begin{aligned}\|x_{t+1} - x^*\|_{H_t}^2 &= (x_t - \tau_t \circ g_t - x^*)^\top H_t(x_t - \tau_t \circ g_t - x^*) \\ &= \|x_t - x^*\|_{H_t}^2 - 2(x_t - x^*)^\top H_t(\tau_t \circ g_t) + \|\tau_t \circ g_t\|_{H_t}^2 \\ &\leq \|x_t - x^*\|_{H_t}^2 - 2\eta(x_t - x^*)^\top F'(x_t,\xi_t) + \|\tau_t \circ g_t\|_{H_t}^2.\end{aligned}$$

The inequality above and the convexity of $F(x,\xi)$ in x implies

$$\begin{aligned}F(x_t,\xi_t) - F(x^*,\xi_t) &\leq \frac{\|x_t - x^*\|_{H_t}^2}{2\eta} - \frac{\|x_{t+1} - x^*\|_{H_t}^2}{2\eta} + \frac{\|\tau_t \circ g_t\|_{H_t}^2}{2\eta} \\ &= \frac{\|x_t - x^*\|_{H_t}^2}{2\eta} - \frac{\|x_{t+1} - x^*\|_{H_t}^2}{2\eta} + \sum_{i=1}^B \frac{\eta\|F_i'(x_t,\xi_t)\|_2^2\|g_t^i\|_{(H_t^i)^{-1}}^2}{2}.\end{aligned}$$

Taking expectation over ξ_t for $t = 1, 2, \ldots$ and averaging the above inequality give

$$\begin{aligned}&\mathbb{E}[f(\bar{x}_T) - f(x^*)] \\ \leq\ & \frac{1}{T}\sum_{t=1}^T\left[\frac{\mathbb{E}\|x_t - x^*\|_{H_t}^2}{2\eta} - \frac{\mathbb{E}\|x_{t+1} - x^*\|_{H_t}^2}{2\eta}\right] + \sum_{t=1}^T\sum_{i=1}^B \frac{\eta\mathbb{E}\left[\|F_i'(x_t,\xi_t)\|_2^2\|g_t^i\|_{(H_t^i)^{-1}}^2\right]}{2T} \\ \leq\ & \frac{1}{T}\sum_{t=1}^T\left[\frac{\mathbb{E}\|x_t - x^*\|_{H_t}^2}{2\eta} - \frac{\mathbb{E}\|x_{t+1} - x^*\|_{H_t}^2}{2\eta}\right] + \sum_{t=1}^T\sum_{i=1}^B \frac{\eta M_i^2\mathbb{E}\left[\|g_t^i\|_{(H_t^i)^{-1}}^2\right]}{2T}, \qquad (4.4.1)\end{aligned}$$

where we use the fact that $\|F_i'(x_t,\xi_t)\|_2^2 \le M_i^2$ in the second inequality.

According to the equation (24) in the proof of Lemma 4 in [47], we have

$$\sum_{t=1}^{T}\|g_t^i\|^2_{(H_t^i)^{-1}} = \sum_{t=1}^{T}\sum_{j=d_1+d_2+\cdots+d_{i-1}+1}^{d_1+d_2+\cdots+d_i}\frac{g_{t,j}^2}{\delta+\|g_{1:t,j}\|_2} \le \sum_{j=d_1+d_2+\cdots+d_{i-1}+1}^{d_1+d_2+\cdots+d_i} 2\|g_{1:T,j}\|_2. \quad (4.4.2)$$

Following the analysis in the proof of Theorem 5 in [47], we show that

$$\begin{aligned}\|x_{t+1}-x^*\|^2_{H_{t+1}} - \|x_{t+1}-x^*\|^2_{H_t} &= \langle x^*-x_{t+1}, \mathrm{diag}(s_{t+1}-s_t)(x^*-x_{t+1})\rangle \\ &\le D_\infty^2\|s_{t+1}-s_t\|_1 = D_\infty^2\langle s_{t+1}-s_t, \mathbf{1}\rangle. \quad (4.4.3)\end{aligned}$$

After applying (4.4.2) and (4.4.3) to (4.4.1) and reorganizing terms, we have

$$\begin{aligned}&\mathbb{E}[f(\bar{x}_T)-f(x^*)] \\ \le\ & \frac{\|x_1-x^*\|^2_{H_1}}{2\eta T} + \frac{D_\infty^2\mathbb{E}\langle s_T,\mathbf{1}\rangle}{2\eta T} + \sum_{i=1}^{B}\frac{\eta\mathbb{E}\left[M_i^2\sum_{j=d_1+d_2+\cdots+d_{i-1}+1}^{d_1+d_2+\cdots+d_i}\|g_{1:T,j}\|_2\right]}{T} \\ \le\ & \frac{\|x_1-x^*\|^2_{H_1}}{2\eta T} + \frac{D_\infty^2\sqrt{Bd}}{2\eta\sqrt{T}} + \sum_{i=1}^{B}\frac{\eta\mathbb{E}\left[M_i^2\sum_{j=d_1+d_2+\cdots+d_{i-1}+1}^{d_1+d_2+\cdots+d_i}\|g_{1:T,j}\|_2\right]}{T},\end{aligned}$$

where the second inequality is because $\langle s_T,\mathbf{1}\rangle = \sum_{j=1}^{d}\|g_{1:T,j}\|_2 \le \sqrt{TBd}$ which holds due to Cauchy-Schwarz inequality and the fact that $\|g_t^i\|_2 = 1$. Then, we obtain the first inequality in the conclusion of the theorem. To obtain the second inequality, we only need to observe that

$$\sum_{j=d_1+d_2+\cdots+d_{i-1}+1}^{d_1+d_2+\cdots+d_i}\|g_{1:T,j}\|_2 \le \sqrt{Td_i} \quad (4.4.4)$$

which holds because of Cauchy-Schwarz inequality and the fact that $\|g_t^i\|_2 = 1$. ■

Remark 1 *When $B = 1$, namely, the normalization is applied to the full gradient instead of different blocks of the gradient, the inequality in Theorem 4.4.1 becomes*

$$\mathbb{E}[f(\bar{x}_T)-f(x^*)] \le \frac{\|x_1-x^*\|^2_{H_1}}{2\eta T} + \frac{D_\infty^2\sqrt{d}}{2\eta\sqrt{T}} + \frac{\eta M^2\sqrt{d}}{\sqrt{T}},$$

where M is a constant such that $\|F'(x_t,\xi_t)\|_2 \le M$. Note that the right hand side of this inequality can be larger than that of the inequality in Theorem 4.4.1 with $B > 1$. We use $B = 2$ as an example. Suppose F_1' dominates in the norm of F', i.e., $M_2 \ll M_1 \approx M$ and $d_1 \ll d_2 \approx d$, we can have $\sum_{i=1}^{B} M_i^2\sqrt{d_i} = O(M^2\sqrt{d_1} + M_2^2\sqrt{d})$ which can be much smaller than the factor $M^2\sqrt{d}$ in the inequality above, especially when M and d are both large. Hence, the optimal value for B is not necessarily one.

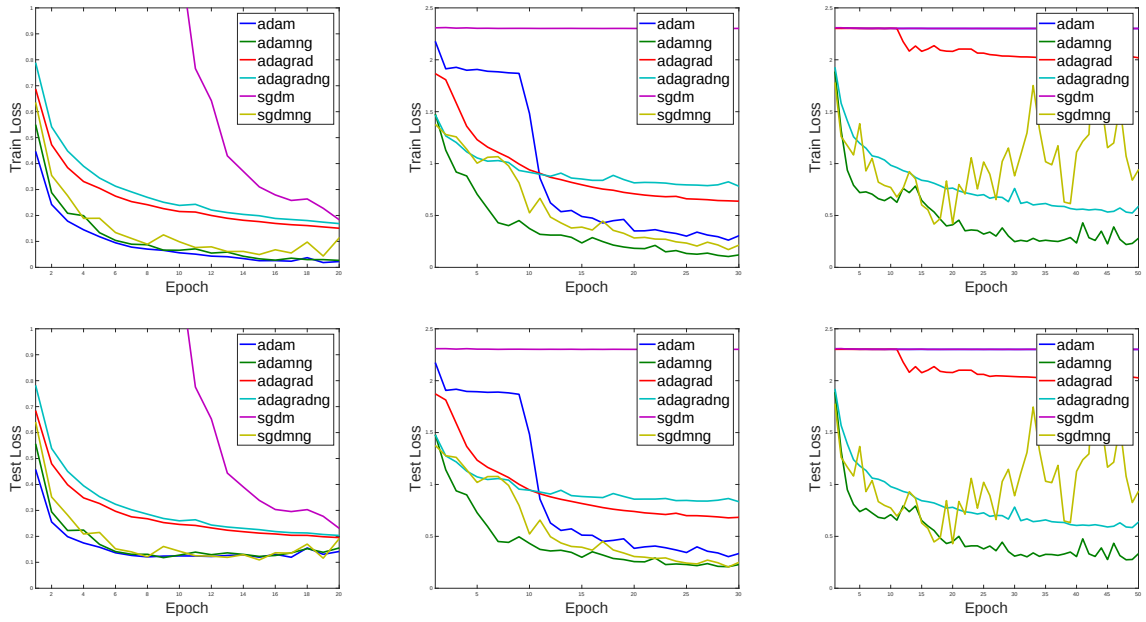

Figure 4.1: The training and testing objective curves on MNIST dataset with multi layer perceptron. From left to right, the layer numbers are 6, 12 and 18 respectively. The first row is the training curve and the second is testing.

4.5 Numerical Experiments

Basic Experiment Setup In this section, we conduct comprehensive numerical experiments on different types of neural networks. The algorithms we are testing are SGD with Momentum (SGDM), AdaGrad [46], Adam [83] and their block-normalized gradient counterparts, which are denoted with suffix "NG". Specifically, we partition the parameters into block as $x = (x^1, x^2, \ldots, x^B)$ such that x^i corresponds to the vector of parameters (including the weight matrix and the bias/intercept coefficients) used in the ith layer in the network.

Our experiments are on four diverse tasks, ranging from image classification to natural language processing. The neural network structures under investigation include multi layer perceptron, long-short term memory and convolution neural networks.

To exclude the potential effect that might be introduced by advanced techniques, in all the experiments, we only adopt the basic version of the neural networks, unless otherwise stated. The loss functions for classifications are cross entropy, while the one for language modeling is log perplexity. Since the computational time is proportional to the epochs, we only show the performance versus epochs. Those with running time are similar so we omit them for brevity. For all the algorithms, we use their default settings. More specifically, for Adam/AdamNG, the initial step size scale $\alpha = 0.001$, first order momentum $\beta_1 = 0.9$, second order momentum $\beta_2 = 0.999$, the parameter to avoid division of zero $\epsilon = 1e^{-8}$; for AdaGrad/AdaGradNG, the initial step size scale is 0.01.

4.5.1 Multi Layer Perceptron for MNIST Image Classification

The first network structure we are going to test upon is the Multi Layer Perceptron (MLP). We will adopt the handwritten digit recognition data set MNIST[2][92], in which, each data is an image of hand written digits from $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 0\}$. There are 60k training and 10k testing examples and the task is to tell the right number contained in the test image. Our approach is applying MLP to learn an end-to-end classifier, where the input is the raw 28×28 images and the output is the label probability. The predicted label is the one with the largest probability. In each middle layer of the MLP, the hidden unit number are 100, and the first and last layer respectively contain 784 and 10 units. The activation functions between layers are all sigmoid and the batch size is 100 for all the algorithms.

We choose different numbers of layer from $\{6, 12, 18\}$. The results are shown in Figure 4.1. Each column of the figures corresponds to the training and testing objective curves of the MLP with a given layer number. From left to right, the layer numbers are respectively 6, 12 and 18. We can see that, when the network is as shallow as containing 6 layers, the normalized stochastic gradient descent can outperform its unnormalized counterpart, while the Adam and AdaGrad are on par with or even slightly better than their unnormalized versions. As the networks become deeper, the acceleration brought by the gradient normalization turns more significant. For example, starting from the second column, AdamNG outperforms Adam in terms of both training and testing convergence. In fact, when the network depth is 18, the AdamNG can still converge to a small objective value while Adam gets stuck from the very beginning. We can observe the similar trend in the comparison between AdaGrad (resp. SGDM) and AdaGradNG (resp. SGDMNG). On the other hand, the algorithms with adaptive step sizes can usually generate a stable learning curve. For example, we can see from the last two column that SGDNG causes significant fluctuation in both training and testing curves. Finally, under any setting, AdamNG is always the best algorithm in terms of convergence performance.

4.5.2 Residual Network on CIFAR10 and CIFAR100

Datasets In this section, we benchmark the methods on CIFAR (both CIFAR10 and CIFAR100) datasets with the residual networks [61], which consist mainly of convolution layers and each layer comes with batch normalization [74]. CIFAR10 consists of 50,000 training images and 10,000 test images from 10 classes, while CIFAR100 from 100 classes. Each input image consists of 32×32 pixels. The dataset is preprocessed as described in [61] by subtracting the means and dividing the variance for each channel. We follow the same data augmentation in [61] that 4 pixels are padded on each side, and a 32×32 crop is randomly sampled from the padded image or its horizontal flip.

Algorithms We adopt two types of optimization frameworks, namely SGD and Adam[3], which respectively represent the constant step size and adaptive step size methods. We compare the performances of their original version and the layer-normalized gradient counterpart. We also

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

[3] We also tried AdaGrad, but it has significantly worse performance than SGD and Adam, so we do not report its result here.

Algorithm	ResNet-20	ResNet-32	ResNet-44	ResNet-56	ResNet-110
			Adam		
Adam	9.14 ± 0.07	8.33 ± 0.17	7.794 ± 0.22	7.33 ± 0.19	6.75 ± 0.30
AdamCLIP	10.18 ± 0.16	9.18 ± 0.06	8.89 ± 0.14	9.24 ± 0.19	9.96 ± 0.29
AdamNG	9.42 ± 0.20	8.50 ± 0.17	8.06 ± 0.20	7.69 ± 0.19	7.29 ± 0.08
$\text{AdamNG}_{\text{adap}}$	**8.52± 0.16**	**7.62± 0.25**	**7.28± 0.18**	**7.04± 0.27**	**6.71± 0.17**
			SGD+Momentum		
SGDM^*	8.75	7.51	7.17	6.97	6.61 ± 0.16
SGDM	7.93 ± 0.15	7.15 ± 0.20	7.09 ± 0.21	7.34 ± 0.52	7.07 ± 0.65
SGDMCLIP	9.03 ± 0.15	8.44 ± 0.14	8.55 ± 0.20	8.30 ± 0.08	8.35 ± 0.25
SGDMNG	7.82 ± 0.26	7.09 ± 0.13	6.60 ± 0.21	6.59 ± 0.23	6.28 ± 0.22
$\text{SGDMNG}_{\text{adap}}$	**7.71 ± 0.18**	**6.90± 0.11**	**6.43± 0.03**	**6.19± 0.11**	**5.87± 0.10**

Table 4.1: Error rates of ResNets with different depths on CIFAR 10. SGDM^* indicates the results reported in [60] with the same experimental setups as ours, where only ResNet-110 has multiple runs.

investigate how the performance changes if the normalization is relaxed to not be strictly 1. In particular, we find that if the normalized gradient is scaled by its variable norm with a ratio, which we call NG_{adap} and defined as follows,

$$\begin{aligned}\text{NG}_{\text{adap}} &:= \text{NG} \times \text{Norm of variable} \times \alpha \\ &= \text{Grad} \times \frac{\text{Norm of variable}}{\text{Norm of grad}} \times \alpha\end{aligned}$$

we can get lower testing error, which has the same finding of an independent and concurrent work [206]. The subscript "adap" is short for "adaptive", as the resulting norm of the gradient is adaptive to its variable norm, while α is the constant ratio. Finally, we also compare with the gradient clipping trick that rescales the gradient norm to a certain value if it is larger than that threshold. Those methods are with suffix "CLIP".

Parameters In the following, whenever we need to tune the parameter, we search the space with a holdout validation set containing 5000 examples.

For SGD+Momentum method, we follow exactly the same experimental protocol as described in [61] and adopt the publicly available Torch implementation[4] for residual network. In particular, SGD is used with momentum of 0.9, weight decay of 0.0001 and mini-batch size of 128. The initial learning rate is 0.1 and dropped by a factor of 0.1 at 80, 120 with a total training of 160 epochs. The weight initialization is the same as [60].

For Adam, we search the initial learning rate in range $\{0.0005, 0.001, 0.005, 0.01\}$ with the base algorithm Adam. We then use the best learning rate, i.e., 0.001, for all the related methods AdamCLIP, AdamNG, and $\text{AdamNG}_{\text{adap}}$. Other setups are the same as the SGD. In particular,

[4] `https://github.com/facebook/fb.resnet.torch`

Algorithm	ResNet-20	ResNet-32	ResNet-44	ResNet-56	ResNet-110
Adam					
Adam	34.44 ± 0.33	32.94 ± 0.16	31.53 ± 0.13	30.80 ± 0.30	28.20 ± 0.14
AdamCLIP	38.10 ± 0.48	35.78 ± 0.20	35.41± 0.19	35.62± 0.39	39.10± 0.35
AdamNG	35.06 ± 0.39	33.78 ± 0.07	32.26± 0.29	31.86 ± 0.21	29.87 ± 0.49
$\text{AdamNG}_{\text{adap}}$	**32.98± 0.52**	**31.74± 0.07**	**30.75± 0.60**	**29.92± 0.26**	**28.09± 0.46**
SGD+Momentum					
SGDM	32.28 ± 0.16	30.62 ± 0.36	29.96 ± 0.66	29.07 ± 0.41	28.79 ± 0.63
SGDMCLIP	35.06 ± 0.37	34.49± 0.49	33.36± 0.36	34.00± 0.96	33.38± 0.73
SGDMNG	32.46 ± 0.37	31.16 ± 0.37	30.05 ± 0.29	29.42 ± 0.51	27.49 ± 0.25
$\text{SGDMNG}_{\text{adap}}$	**31.43 ± 0.35**	**29.56 ± 0.25**	**28.92 ± 0.28**	**28.48 ± 0.19**	**26.72 ± 0.39**

Table 4.2: Error rates of ResNets with different depths on CIFAR 100. Note that [60] did not run experiment on CIFAR 100.

we also adopt the manually learning rate decay here, since, otherwise, the performance will be much worse.

We adopt the residual network architectures with depths $L = \{20, 32, 44, 56, 110\}$ on both CIFAR-10 and CIFAR100. For the extra hyper-parameters, i.e., threshold of gradient clipping and scale ratio of NG_{adap}, i.e., α, we choose the best hyper-parameter from the 56-layer residual network. In particular, for clipping, the searched values are $\{0.05, 0.1, 0.5, 1, 5\}$, and the best value is 0.1. For the ratio α, the searched values are $\{0.01, 0.02, 0.05\}$ and the best is 0.02.

Results For each network structure, we make 5 runs with random initialization. We report the training and testing curves on CIFAR10 and CIFAR100 datasets with deepest network Res-110 in Figure 4.2. We can see that the normalized gradient methods (with suffix "NG") converge the fastest in training, compared to the non-normalized counterparts. While the adaptive version NG_{adap} is not as fast as NG in training, however, it can always converge to a solution with lower testing error, which can be seen more clearly in Table 4.1 and 4.2.

For further quantitative analysis, we report the means and variances of the final test errors on both datasets in Table 4.1 and 4.2, respectively. Both figures convey the following two messages. Firstly, on both datasets with ResNet, SGD+Momentum is uniformly better than Adam in that it always converges to a solution with lower test error. Such advantage can be immediately seen by comparing the two row blocks within each column of both tables. It is also consistent with the common wisdom [197]. Secondly, for both Adam and SGD+Momentum, the NG_{adap} version has the best generalization performance (which is mark bold in each column), while the gradient clipping is inferior to all the remaining variants. While the normalized SGD+Momentum is better than the vanilla version, Adam slightly outperforms its normalized counterpart. Those observations are consistent across networks with variant depths.

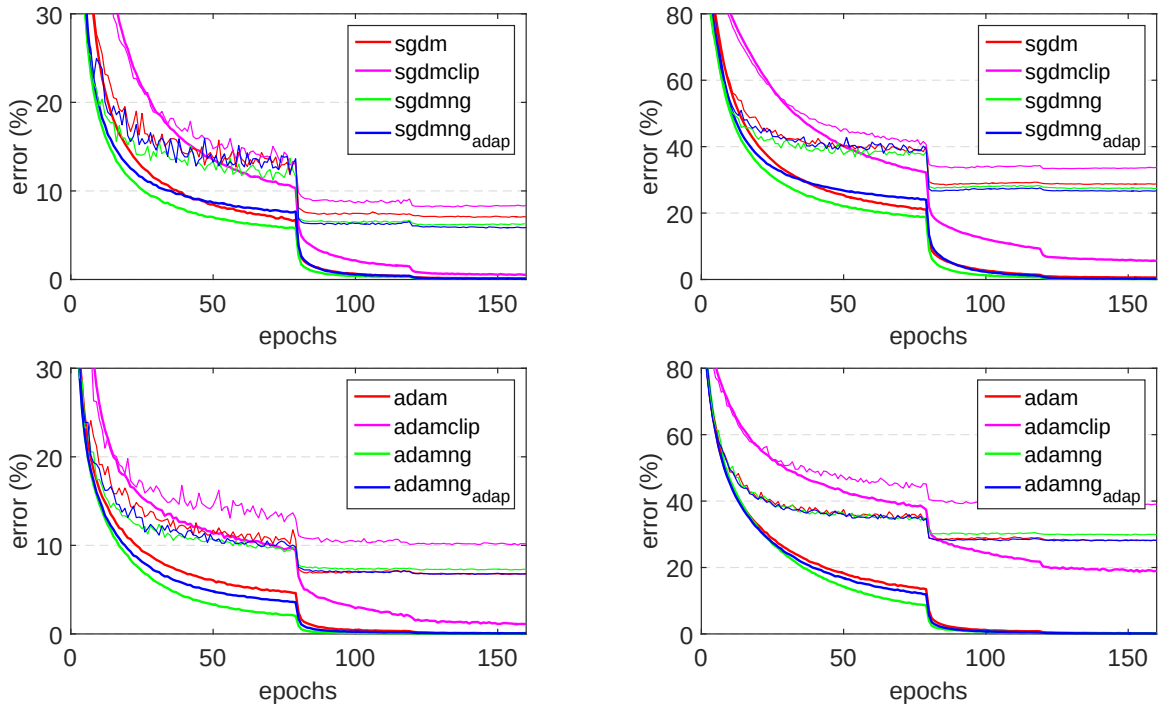

Figure 4.2: The training and testing curves on CIFAR10 and CIFAR100 datasets with Resnet-110. Left: CIFAR10; Right: CIFAR100; Upper: SGD+Momentum; Lower: Adam. The thick curves are the training while the thin are testing.

4.5.3 Residual Network for ImageNet Classification

In this section, we further test our methods on ImageNet 2012 classification challenges, which consists of more than 1.2M images from 1,000 classes. We use the given 1.28M labeled images for training and the validation set with 50k images for testing. We employ the validation set as the test set, and evaluate the classification performance based on top-1 and top-5 error. The pre-activation [62] version of ResNet is adopted in our experiments to perform the classification task. Like the previous experiment, we again compare the performance on SGD+Momentum and Adam.

We run our experiments on one GPU and use single scale and single crop test for simplifying discussion. We keep all the experiments settings the same as the publicly available Torch implementation [5]. That is, we apply stochastic gradient descent with momentum of 0.9, weight decay of 0.0001, and set the initial learning rate to 0.1. The exception is that we use mini-batch size of 64 and 50 training epochs considering the GPU memory limitations and training time costs. Regarding learning rate annealing, we use 0.001 exponential decay.

As for Adam, we search the initial learning rate in range $\{0.0005, 0.001, 0.005, 0.01\}$. Other setups are the same as the SGD optimization framework. Due to the time-consuming nature of training the networks (which usually takes one week) in this experiment, we only test on a 34-

[5]We again use the public Torch implementation: `https://github.com/facebook/fb.resnet.torch`

layer ResNet and compare SGD and Adam with our default NG method on the testing error of the classification. From Table 3, we can see normalized gradient has a non-trivial improvement on the testing error over the baselines SGD and Adam. Besides, the SGD+Momentum again outperforms Adam, which is consistent with both the common wisdom [197] and also the findings in previous section.

method	Top-1	Top-5
Adam	35.6	14.09
AdamNG	30.17	10.51
SGDM	29.05	9.95
SGDMNG	**28.43**	**9.57**

Table 4.3: Top-1 and Top 5 error rates of ResNet on ImageNet classification with different algorithms.

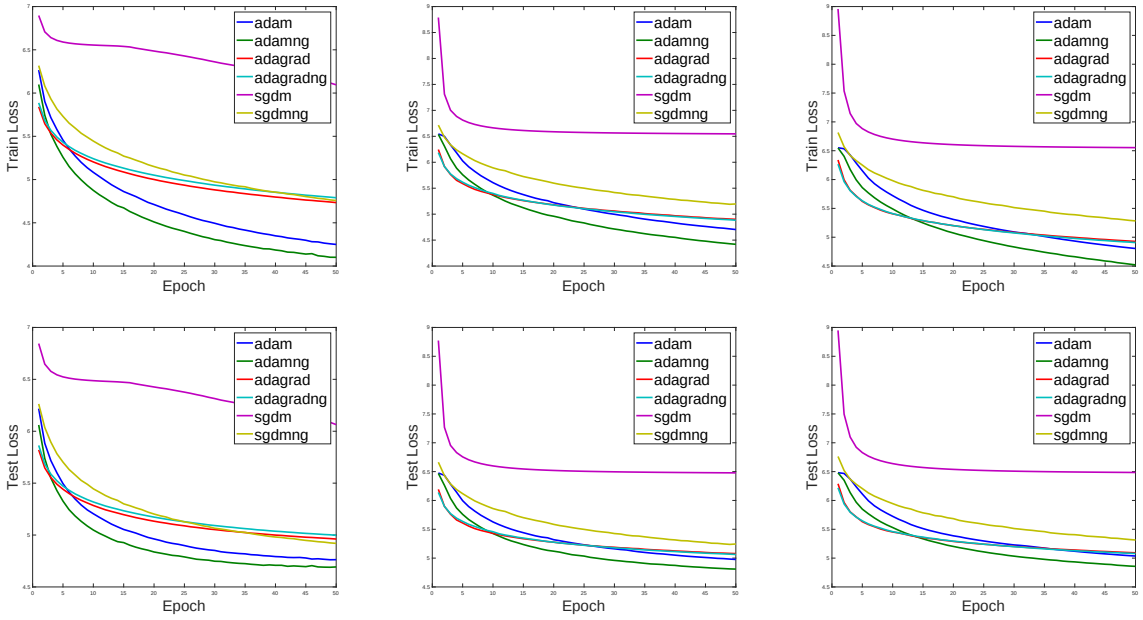

Figure 4.3: The training and testing objective curves on Penn Tree Bank dataset with LSTM recurrent neural networks. The first row is the training objective while the second is the testing. From left to right, the training sequence (BPTT) length are respectively 40, 400 and 1000. Dropout with 0.5 is imposed.

4.5.4 Language Modeling with Recurrent Neural Network

Now we move on to test the algorithms on Recurrent Neural Networks (RNN). In this section, we test the performance of the proposed algorithm on the word-level language modeling task with a

Algorithm	AdamNG	Adam	AdaGradNG	AdaGrad	SGDMNG	SGDM
Validation Accuracy	**77.11%**	74.02%	71.95%	69.89%	71.95%	64.35%

Table 4.4: The Best validation accuracy achieved by the different algorithms.

popular type of RNN, i.e. single directional Long-Short Term Memory (LSTM) networks [69]. The data set under use is Penn Tree Bank (PTB) [115] data, which, after preprocessed, contains 929k training words, 73k validation and 82k test words. The vocabulary size is about 10k. The LSTM has 2 layers, each containing 200 hidden units. The word embedding has 200 dimensions which is trained from scratch. The batch size is 100. We vary the length of the backprop through time (BPTT) within the range $\{40, 400, 1000\}$. To prevent overfitting, we add a dropout regularization with rate 0.5 under all the settings.

The results are shown in Figure 4.3. The conclusions drawn from those figures are again similar to those in the last two experiments. However, the slightly different yet cheering observations is that the AdamNG is uniformly better than all the other competitors with any training sequence length. The superiority in terms of convergence speedup exists in both training and testing.

4.5.5 Sentiment Analysis with Convolution Neural Network

The task in this section is the sentiment analysis with convolution neural network. The dataset under use is Rotten Tomatoes[6] [137], a movie review dataset containing 10,662 documents, with half positive and half negative. We randomly select around 90% for training and 10% for validation. The model is a single layer convolution neural network that follows the setup of [82]. The word embedding under use is randomly initialized and of 128-dimension.

For each algorithm, we run 150 epochs on the training data, and report the best validation accuracy in Table 4.4. The messages conveyed by the table is three-fold. Firstly, the algorithms using normalized gradient achieve much better validation accuracy than their unnormalized versions. Secondly, those with adaptive stepsize always obtain better accuracy than those without. This is easily seen by the comparison between Adam and SGDM. The last point is the direct conclusion from the previous two that the algorithm using normalized gradient with adaptive step sizes, namely AdamNG, outperforms all the remaining competitors.

4.6 Discussion

In this chapter, we propose a generic algorithm framework for first order optimization. Our goal is to provide a simple alternative to train deep neural networks. It is particularly effective for addressing the vanishing and exploding gradient challenge in training with non-convex loss functions, such as in the context of convolutional and recurrent neural networks. Our method is based on normalizing the gradient to establish the descending direction regardless of its magnitude, and then separately estimating the ideal step size adaptively or constantly. This method is

[6] `http://www.cs.cornell.edu/people/pabo/movie-review-data/`

quite general and may be applied to different types of networks and various architectures. Although the primary application of the algorithm is deep neural network training, we provide a convergence for the new method under the convex setting.

Empirically, the proposed method exhibits very promising performance in training different types of networks (convolutional, recurrent) across multiple well-known data sets (image classification, natural language processing, sentiment analysis, etc.). In general, the positive performance differential compared to the baselines is most striking for very deep networks, as shown in our comprehensive experimental study.

Chapter 5

DSPDC: Doubly Stochastic Primal-Dual Coordinate Method for Bilinear Saddle-Point Problem

5.1 Introduction

In this chapter, we consider regularized *empirical risk minimization* (ERM) problems of the following form:

$$\min_{x\in\mathbb{R}^p}\left\{P(x)\equiv\frac{1}{n}\sum_{i=1}^{n}\phi_i(a_i^Tx)+g(x)\right\}, \tag{5.1.1}$$

where $a_1,\ldots,a_n\in\mathbb{R}^p$ are n data points with p features, $\phi_i:\mathbb{R}\to\mathbb{R}$ is a convex loss function of the linear predictor a_i^Tx, for $i=1,\ldots,n$, and $g:\mathbb{R}^p\to\mathbb{R}$ is a convex regularization function for the coefficient vector $x\in\mathbb{R}^p$ in the linear predictor. We assume g has a *decomposable structure*, namely,

$$g(x)=\sum_{j=1}^{p}g_j(x_j), \tag{5.1.2}$$

where $g_j:\mathbb{R}\to\mathbb{R}$ is only a function of x_j, the j-th coordinate of x. For simplicity, we consider a univariate g_j at this moment. In Section 5.5, the proposed method will be generalized for the problems having a block-wise decomposable structure with multivariate g_j. We further make the following assumptions:

Assumption 1 *For any $\alpha,\beta\in\mathbb{R}$,*

- *g_j is λ-strongly convex for $j=1,2,\ldots,p$, i.e., $g_j(\alpha)\geq g_j(\beta)+g_j'(\beta)(\alpha-\beta)+\frac{\lambda}{2}(\alpha-\beta)^2$;*
- *ϕ_i is $(1/\gamma)$-smooth for $i=1,2,\ldots,n$, i.e., $\phi_i(\alpha)\leq\phi_i(\beta)+\nabla\phi_i(\beta)(\alpha-\beta)+\frac{1}{2\gamma}(\alpha-\beta)^2$.*

The problem (5.1.1) captures many applications in business analytics, statistics, machine learning and data mining, and has triggered many studies in the optimization community. Typically, for each data point a_i, there is an associated response value $b_i\in\mathbb{R}$, which can be continuous

(in regression problems) or discrete (in classification problems). The examples of loss function $\phi_i(\cdot)$ associated to (a_i, b_i) include:

- *Square Loss*, where $a_i \in \mathbb{R}^p$, $b_i \in \mathbb{R}$ and $\phi_i(z) = \frac{1}{2}(z - b_i)^2$, which corresponds to *linear regression* problem;
- *Sigmoid Loss*, where $a_i \in \mathbb{R}^p$, $b_i \in \{1, -1\}$ and $\phi_i(z) = \log(1 + \exp(-b_i z))$, which corresponds to *logistic regression* problem;
- *Smooth Hinge Loss*, where $a_i \in \mathbb{R}^p$, $b_i \in \{1, -1\}$ and

$$\phi_i(z) = \begin{cases} 0 & \text{if } b_i z \geq 1 \\ \frac{1}{2} - b_i z & \text{if } b_i z \leq 0 \\ \frac{1}{2}(1 - b_i z)^2 & \text{otherwise.} \end{cases} \tag{5.1.3}$$

  which corresponds to the smooth *support vector machine* problem.

In fact, if appropriate reformulation is conducted, many other problems can also be reduced to (5.1.1), for example, the *multi-task large margin nearest neighbor metric learning* (MT-LMNN) problem (See Section 5.5.3).

The commonly used regularization terms include the ℓ_2-regularization $g_j(x) = \frac{\lambda x^2}{2}$ with $\lambda > 0$ and $\ell_2 + \ell_1$-regularization $g_j(x) = \frac{\lambda_2 x^2}{2} + \lambda_1 |x|$ with $\lambda_1, \lambda_2 > 0$.

We often call (5.1.1) the *primal problem* and its conjugate *dual problem* is

$$\max_{y \in \mathbb{R}^n} \left\{ D(y) \equiv -g^* \left(-\frac{A^T y}{n} \right) - \frac{1}{n} \sum_{i=1}^n \phi_i^*(y_i) \right\}, \tag{5.1.4}$$

where $A = [a_1, a_2, \ldots, a_n]^T \in \mathbb{R}^{n \times p}$ and ϕ_i^* and g^* are the convex conjugates of ϕ_i and g, respectively, meaning that $g^*(v) = \max_{u \in \mathbb{R}^p} \langle u, v \rangle - g(u)$ and $\phi_i^*(\alpha) = \max_{\beta \in \mathbb{R}} \alpha\beta - \phi_i(\beta)$. It is well-known in convex analysis that, under Assumption 1, g^* is $\frac{1}{\lambda}$-smooth and ϕ_i^* is γ-strongly convex. In this chapter, instead of considering purely (5.1.1) or (5.1.4), we are interested in their associated *saddle-point* problem:

$$\min_{x \in \mathbb{R}^p} \max_{y \in \mathbb{R}^n} \left\{ g(x) + \frac{1}{n} y^T A x - \frac{1}{n} \sum_{i=1}^n \phi_i^*(y_i) \right\}. \tag{5.1.5}$$

Let $x^\star$ and $y^\star$ be the optimal solutions of (5.1.1) and (5.1.4), respectively. It is known that the pair $(x^\star, y^\star)$ is a *saddle point* of (5.1.5) in the sense that

$$x^\star = \operatorname*{argmin}_{x \in \mathbb{R}^p} \left\{ g(x) + \frac{1}{n} (y^\star)^T A x - \frac{1}{n} \sum_{i=1}^n \phi_i^*(y_i^\star) \right\}, \tag{5.1.6}$$

$$y^\star = \operatorname*{argmax}_{y \in \mathbb{R}^n} \left\{ g(x^\star) + \frac{1}{n} y^T A x^\star - \frac{1}{n} \sum_{i=1}^n \phi_i^*(y_i) \right\}. \tag{5.1.7}$$

The contributions of this chapter can be highlighted as follow:

- We propose a *doubly stochastic primal-dual coordinate* (DSPDC) method for solving problem (5.1.5) that randomly samples q out of p primal and m out of n dual coordinates to update in each iteration.

- We show that DSPDC method generates a sequence of primal-dual iterates that *linearly* converges to $(x^\star, y^\star)$ and the primal-dual objective gap along this sequence also *linearly* converges to zero.
- We generalize this approach to bilinear saddle-point problems with a block-wise decomposable structure, and show a similar iteration complexity for finding an ϵ-optimal solution.
- We show that the proposed method has a lower *overall complexity* than existing coordinate methods when either the data matrix has a factorized structure or the proximal mapping on each block is computationally expensive, e.g., involving an eigenvalue decomposition.
- Our experiments confirm the efficiency of DSPDC on both synthetic and real datasets in various scenarios. A notable application is the *multi-task large margin nearest neighbor* (MT-LMNN) metric learning problem.

Notation Before presenting our approach, we first introduce the notations that will be used throughout the chapter. Let $[d]$ represent the set $\{1, 2, ..., d\}$. For $v \in \mathbb{R}^d$, let v_i be its i-th coordinate for $i \in [d]$ and v_I be a sub-vector of v that consists of the coordinates of v indexed by a set $I \subset [d]$. Given an $n \times p$ matrix W, we denote its i-th row and j-th column by W_i and W^j, respectively. For $I \subset [n]$ and $J \subset [p]$, the matrices W_I and W^J represent sub-matrices of W that consist of the rows indexed by I and columns indexed by J, respectively. We denote the entry of W in i-th row and j-th column by W_i^j and let W_I^J be sub-matrix of W where the rows indexed by I intersect with the columns indexed by J.

Let $\langle \cdot, \cdot \rangle$ be the inner product in a Euclidean space, $\| \cdot \|$ be the ℓ_2-norm of a vector and $\| \cdot \|_2$ and $\| \cdot \|_F$ be the spectral norm and the Frobenius norm of a matrix, respectively. For integers $q \in [p]$ and $m \in [n]$, we define $\Lambda_{q,m}$ as a *scale constant* of the data as follows

$$\Lambda_{q,m} \equiv \max_{I \subset [n], J \subset [p], |I|=m, |J|=q} \|A_I^J\|_2^2. \tag{5.1.8}$$

The maximum ℓ_2 norm of data points is therefore $\sqrt{\Lambda_{p,1}}$. The *condition number* of problems (5.1.1),(5.1.4), and (5.1.5) is usually defined as

$$\kappa \equiv \frac{\Lambda_{p,1}}{\lambda\gamma}, \tag{5.1.9}$$

which affects the iteration complexity of many first-order methods.

5.2 Related Work

To find an ϵ-optimal solution of problem (5.1.1), (5.1.4) or (5.1.5), the *overall complexity* of an iterative method is defined as the per-iteration computational cost multiplied by the total number of required iterations (called *iteration complexity*). Deterministic first-order methods such as Chambolle and Pock [18], Nemirovski [126], Nesterov [128, 130], Yu et al. [207] have to compute a full gradient in each iteration by going through all p features of all n instances at a per-iteration cost of $O(np)$, which can be inefficient for big data. Therefore, stochastic optimization methods that sample one instance or one feature in each iteration become more

popular. There are two major categories of stochastic optimization algorithms that are studied actively in recent years: stochastic gradient methods and stochastic coordinate methods. The DSPDC method we propose belongs to the second category.

Recently, there have been increasing interests in *stochastic variance reduced gradient* (SVRG) methods [3, 78, 85, 134, 200]. SVRG runs in multiple stages. At each stage, it computes a full gradient and then performs $O(\kappa)$ iterative updates with stochastic gradients constructed by sampled instances. Since the full gradient is computed only once in each stage, SVRG has a per-iteration cost of $O(p)$, which is lower than deterministic gradient methods, and it needs $O((n+\kappa)\log(1/\epsilon))$ iterations to find an ϵ-optimal solution for problem (5.1.1), so that the overall complexity of SVRG is $O((np+\kappa p)\log(1/\epsilon))$. Recently, an accelerated SVRG method, named Katyusha [3], further reduces the iteration complexity of SVRG to $O((n+\sqrt{n\kappa})\log(1/\epsilon))$ while maintains the $O(p)$ per-iteration cost so that it achieves an overall complexity of $O((np+\sqrt{n\kappa}p)\log(1/\epsilon))$. The aforementioned overall complexities are obtained when a *uniform sampling* scheme is applied in the construction of stochastic gradient. One can further reduce the κ term in these complexities by using a *non-uniform sampling* scheme as pointed out, for example, by Xiao and Zhang [200]. However, in this paper, the complexity of each algorithm we present and compare is based on a uniform sampling scheme unless stated otherwise. After the earlier version of our draft [209] was posted online[1], Balamurugan and Bach [12] developed an accelerated SVRG method (ASVRG-SP) for solving the saddle-point formulation (5.1.5), which has a complexity[2] of $\tilde{O}((np+np\sqrt{\frac{\max\{\Lambda_{p,1},\Lambda_{1,n}\}}{\lambda\gamma}})\log(1/\epsilon))$ by uniform sampling and $\tilde{O}((np+\sqrt{np}\sqrt{\frac{\max\{n,p\}\|A\|_F^2}{\lambda\gamma}})\log(1/\epsilon))$ by non-uniform sampling. Here and in the rest of the paper, $\tilde{O}$ contains some logarithmic factors.

Stochastic incremental gradient methods [12, 39, 40, 89, 91, 113, 161] is also widely studied in recent literature. Different from SVRG, stochastic incremental gradient method computes a full gradient only once at the beginning, but maintains and updates the average of historical stochastic gradients using one sampled instance per iteration. Standard stochastic incremental gradient methods [39, 40, 91, 113, 161] have a per-iteration cost of $O(p)$ just as SVRG and need $O((n+\kappa)\log(1/\epsilon))$ iterations to find an ϵ-optimal solution so that their overall complexity is the same as SVRG. Moreover, an accelerated stochastic incremental gradients method, named RPDG [89], achieves an iteration complexity of only $O((n+\sqrt{n\kappa})\log(1/\epsilon))$ and a per-iteration cost of $O(p)$ so that its overall complexity is the same as Katyusha. The iteration complexity of RPDF and Katyusha is proved to be optimal by Lan and Zhou [89].

In contrast to stochastic gradient methods, *stochastic coordinate* method works by updating randomly sampled coordinates of decision variables [5, 36, 41, 50, 103, 106, 129, 132, 144, 145, 152, 154, 167]. Shalev-Shwartz and Zhang [166, 166, 168] proposed a stochastic dual coordinate ascent (SDCA) method to solve the dual formulation (5.1.4). SDCA has an iteration complexity of $O((n+\kappa)\log(1/\epsilon))$ and has been further improved to the accelerated SDCA (ASDCA) method [168] that achieves an iteration complexity of $\tilde{O}((n+\sqrt{n\kappa})\log(1/\epsilon))$. The optimal iteration complexity $O((n+\sqrt{n\kappa})\log(1/\epsilon))$ is obtained by the accelerated proximal coordinate gradient (APCG) method [103] when it is applied to the dual problem (5.1.4). Extending the

[1] `https://arxiv.org/abs/1508.03390`

[2] This complexity is achieved by the individual-split version of ASVRG-SP.

deterministic algorithm by Chambolle and Pock [18] for saddle-point problems, Zhang and Xiao [220] recently proposed a stochastic primal-dual coordinate (SPDC) method for (5.1.5), which alternates between maximizing over a randomly chosen dual variable and minimizing over all primal variables and also achieves the optimal $O((n+\sqrt{n\kappa})\log(1/\epsilon))$ iteration complexity. The per-iteration cost is $O(p)$ in all of these coordinate methods. Note that, when applied to the primal problem (5.1.1), APCG samples a feature of data in each iterative update and find an ϵ-optimal solution with a per-iteration cost of $O(n)$ in $O((p+p\sqrt{\frac{\Lambda_{1,n}}{n\lambda\gamma}})\log(1/\epsilon))$ iterations, which is also optimal according to [89].

Some recent works [35, 37, 41, 86, 116, 222] made attempts in combining stochastic gradient and stochastic coordinate. Dang and Lan [37], Matsushima et al. [116], Zhao et al. [222] proposed randomized block coordinate methods, which utilize stochastic partial gradient of the selected block based on randomly sampled instances and features in each iteration. However, these methods face a constant variance of stochastic partial gradient so that they need $O(1/\epsilon)$ iterations. These techniques are further improved in Konecný et al. [86], Zhao et al. [222] with the stochastic variance reduced partial gradient but only obtain the sub-optimal $O((n+\kappa)\log(1/\epsilon))$ iteration complexity.

5.3 Summary of Results

Although the aforementioned stochastic coordinate methods have achieved great performances on ERM problem (5.1.5), they either only sample over primal coordinates or only sample over dual coordinates to update in each iteration. Therefore, it is natural to ask the following questions.

- *What is the iteration complexity of a coordinate method for problem (5.1.5) that samples both primal and dual coordinates to update in each iteration?*
- *When is this type of algorithm has a lower overall complexity than purely primal and purely dual coordinate methods?*

To contribute to the answers to these questions, we propose the DSPDC method in Section 5.4 that samples over both features and instances of dataset by randomly choosing the associated primal and dual coordinates to update in each iteration.

To answer the first question, we show in Theorem 5.4.1 and 5.4.2 that, if q primal and m dual coordinates are uniformly sampled and updated in each iteration, the number of iterations DSPDC needs to find an ϵ-optimal solution for (5.1.5) is $O((\sqrt{\frac{\Lambda_{q,m}}{\lambda\gamma n}}\frac{np}{mq}+\max\{\frac{n}{m},\frac{p}{q}\})\log(1/\epsilon))$. This iteration complexity is interesting since it matches the optimal $O((n+\sqrt{n\kappa})\log(1/\epsilon))$ iteration complexity of dual coordinate methods [103, 168, 220] when $(q,m)=(p,1)$, and also matches the optimal $O((p+p\sqrt{\frac{\Lambda_{1,n}}{n\lambda\gamma}})\log(1/\epsilon))$ iteration complexity of primal coordinate methods [103, 106] when $(q,m)=(1,n)$. In Section 5.5, we further generalize DSPDC and its complexity to a bilinear saddle-point problem with a block-wise decomposable structure.

To study the second question, we compare different coordinate algorithms based on the overall complexity for finding an ϵ-optimal solution. For most ERM problems, the per-iteration cost of SPDC is $O(p)$ and its the overall complexity is $O(np+p\sqrt{n\kappa}\log(1/\epsilon))$. When $(q,m)=(1,1)$ and without any assumptions on the sparsity of data, the per-iteration cost of DSPDC is

$O(\min\{n,p\})$ due to a full-dimensional inner product in the algorithm. If $n \geq p$, which is true for most ERM problems, the overall complexity of DSPDC becomes $O((np + \sqrt{\frac{n\Lambda_{1,1}}{\lambda\gamma}}p^2)\log(1/\epsilon))$, which is not lower than that of SPDC in general[3]. Nevertheless, we identify two important cases where DSPDC has a lower overall complexity than SPDC and other existing coordinate methods.

The first case is when data A has a *factorized structure*, namely, $A = UV$ with $U \in \mathbb{R}^{n\times d}$, $V \in \mathbb{R}^{d\times p}$ and $d < \min\{n,p\}$. The ERM problem with factorized data arises when (random) dimension/instance reduction or matrix sketching/factorization techniques are applied to A in order to reduce the storage and computational cost. More examples are provided in Section 5.4.2. In this case, choosing $(q,m) = (1,1)$ and using an efficient implementation, our DSPDC has an overall complexity of $O((nd + \sqrt{\frac{n\Lambda_{1,1}}{\lambda\gamma}}pd)\log(1/\epsilon))$, better than the $O((npd + \sqrt{\kappa n}pd)\log(1/\epsilon))$ complexity of SPDC with the same efficient implementation. See Table 5.1 for comparisons with more existing techniques for this class of problems.

The second case is when solving a block-wise decomposable bilinear saddle-point problem where the proximal mapping on each block is computationally expensive. The applications with this property include trace regression [175] and distance metric learning [138, 191, 192], where each block of variables needs to be a $d \times d$ positive semi-definite matrix so that the proximal mapping involves an eigenvalue decomposition with a complexity of $O(d^3)$. When $(q,m) = (1,1)$ and $n \geq p$, DSPDC requires solving eigenvalue decomposition only for one block of variables so that its overall complexity is $O((d^3 + pd^2)(n + \sqrt{\frac{n\Lambda_{1,1}}{\lambda\gamma}}p)\log(1/\epsilon))$ as shown in Section 5.5, which is lower than the $O((npd^3 + \sqrt{n\kappa}pd^3)\log(1/\epsilon))$ overall complexity of SPDC when $\sqrt{\Lambda_{1,1}}p \leq \sqrt{\Lambda_{p,1}}d$. See Table 5.2 for comparisons with more existing techniques for this class of problems.

Although it is not our main focus, we note that applying a non-uniform sampling on the primal and dual coordinates can further reduce the overall complexity of our DSPDC just as other coordinate methods [5, 31, 32, 144, 145, 154, 220] .

5.4 Doubly Stochastic Primal-Dual Coordinate Method

5.4.1 Algorithm and Convergence Properties

In this section, we propose the doubly stochastic primal-dual coordinate (DSPDC) method in Algorithm 3 for problem (5.1.5). In Algorithm 3, the primal and dual solutions $(x^{(t+1)}, y^{(t+1)})$ are updated as (5.4.3) and (5.4.1) in the randomly selected q and m coordinates indexed by J and I, respectively[4]. These updates utilize the first-order information provided by the vectors $A\bar{x}^{(t)}$ and $A^T\bar{y}^{(t+1)}$ where $(\bar{x}^{(t)}, \bar{y}^{(t+1)})$ are updated using the momentum steps (5.4.4) and (5.4.2) which are commonly used to accelerate gradient (AG) methods [128, 130]. Algorithm 3 requires three control parameters θ, τ and σ and its convergence is ensured after a proper choice of these parameters as shown in Theorem 5.4.1. The proofs of all theorems are deferred to the Appendix.

[3]Note that $\Lambda_{p,1} \geq \Lambda_{1,1} \geq \frac{\Lambda_{p,1}}{p}$ and $\kappa \geq \frac{\Lambda_{1,1}}{\lambda\gamma} \geq \frac{\kappa}{p}$.

[4]Here, we hide the dependency of I and J on t to simplify the notation.

Algorithm 3 Doubly Stochastic Primal-Dual Coordinate (DSPDC) Method

Input: $x^{(-1)} = x^{(0)} = \bar{x}^{(0)} \in \mathbb{R}^p$, $y^{(-1)} = y^{(0)} = \bar{y}^{(0)} \in \mathbb{R}^n$, and parameters (θ, τ, σ).
Output: $x^{(T)}$ and $y^{(T)}$

1: **for** $t = 0, 1, 2, \ldots, T-1$ **do**
2: Uniformly and randomly choose $I \subset [n]$ and $J \subset [p]$ of sizes m and q, respectively.
3: Update the primal and dual coordinates

$$y_i^{(t+1)} = \begin{cases} \operatorname{argmax}_{\beta \in \mathbb{R}} \left\{ \frac{1}{n} \langle A_i, \bar{x}^{(t)} \rangle \beta - \frac{\phi_i^*(\beta)}{n} - \frac{1}{2\sigma} (\beta - y_i^{(t)})^2 \right\} & \text{if } i \in I, \\ y_i^{(t)} & \text{if } i \notin I, \end{cases} \tag{5.4.1}$$

$$\bar{y}^{(t+1)} = y^{(t)} + \frac{n}{m}(y^{(t+1)} - y^{(t)}), \tag{5.4.2}$$

$$x_j^{(t+1)} = \begin{cases} \operatorname{argmin}_{\alpha \in \mathbb{R}} \left\{ \frac{1}{n} \langle A^j, \bar{y}^{(t+1)} \rangle \alpha + g_j(\alpha) + \frac{1}{2\tau} (\alpha - x_j^{(t)})^2 \right\} & \text{if } j \in J, \\ x_j^{(t)} & \text{if } j \notin J, \end{cases} \tag{5.4.3}$$

$$\bar{x}^{(t+1)} = x^{(t)} + (\theta + 1)(x^{(t+1)} - x^{(t)}). \tag{5.4.4}$$

4: **end for**

Theorem 5.4.1 *Suppose θ, τ and σ in Algorithm 3 are chosen so that*

$$\theta = \frac{p}{q} - \frac{p/q}{\sqrt{\frac{\Lambda}{\lambda \gamma n} \frac{np}{mq}} + \max\{\frac{n}{m}, \frac{p}{q}\}}, \ \tau\sigma = \frac{nmq}{4p\Lambda}, \ \frac{p}{2q\lambda\tau} + \frac{p}{q} = \frac{n^2}{2m\gamma\sigma} + \frac{n}{m} \tag{5.4.5}$$

where Λ is any constant such that $\Lambda \geq \Lambda_{q,m}$. For each $t \geq 0$, Algorithm 3 guarantees

$$\begin{aligned} & \left(\frac{p}{2q\tau} + \frac{p\lambda}{q} \right) \mathbb{E} \|x^\star - x^{(t)}\|^2 + \left(\frac{n}{4m\sigma} + \frac{\gamma}{m} \right) \mathbb{E} \|y^\star - y^{(t)}\|^2 \\ \leq & \left(1 - \frac{1}{\max\left\{ \frac{p}{q}, \frac{n}{m} \right\} + \sqrt{\frac{\Lambda}{\lambda\gamma n} \frac{np}{mq}}} \right)^t \left[\left(\frac{p}{2q\tau} + \frac{p\lambda}{q} \right) \|x^\star - x^{(0)}\|^2 + \left(\frac{n}{2m\sigma} + \frac{\gamma}{m} \right) \|y^\star - y^{(0)}\|^2 \right]. \end{aligned}$$

Remark 1 *For a given Λ, the values of τ and σ can be solved from the last two equations of (5.4.5) in closed forms:*

$$\tau = \frac{p}{q\lambda} \left(\left(\frac{n}{m} - \frac{p}{q} \right) + \sqrt{\left(\frac{n}{m} - \frac{p}{q} \right)^2 + \frac{4(np)^2 \Lambda}{(mq)^2 n \lambda \gamma}} \right)^{-1}, \tag{5.4.6}$$

$$\sigma = \frac{n^2}{m\gamma} \left(\left(\frac{p}{q} - \frac{n}{m} \right) + \sqrt{\left(\frac{n}{m} - \frac{p}{q} \right)^2 + \frac{4(np)^2 \Lambda}{(mq)^2 n \lambda \gamma}} \right)^{-1}, \tag{5.4.7}$$

which are referred to as the primal and dual step size, respectively. If both primal and dual coordinates are sampled at the same ratio, i.e., $\frac{q}{p} = \frac{m}{n}$, then we have the following simplified

version:

$$\tau = \frac{m}{2}\sqrt{\frac{\gamma}{\Lambda n \lambda}}, \quad \sigma = \frac{m}{2}\sqrt{\frac{n\lambda}{\Lambda\gamma}}. \tag{5.4.8}$$

According to the convergence rate above, the best choice of Λ is $\Lambda_{q,m}$. Although the exact computation of $\Lambda_{q,m}$ by definition (5.1.8) may be costly, for instance, when $q \approx \frac{p}{2}$ or $m \approx \frac{n}{2}$, it is tractable when q and m are close to 1 or close to p and n. In practice, we suggest choosing $\Lambda = \frac{mqR^2\Lambda_{p,1}}{p}$ as an approximation of $\Lambda_{q,m}$, which provides reasonably good empirical performance (see Section 5.6).

Besides the distance to the saddle-point $(x^\star, y^\star)$, a useful quality measure for the solution $(x^{(t)}, y^{(t)})$ is its primal-dual objective gap, $P(x^{(t)}) - D(y^{(t)})$, because it can be evaluated in each iteration and used as a stopping criterion in practice. The next theorem establishes the convergence rate of the primal-dual objective gap ensured by DSPDC.

Theorem 5.4.2 *Suppose τ and σ are chosen as (5.4.5) while θ is replaced by*

$$\theta = \frac{p}{q} - \frac{p/q}{2\sqrt{\frac{\Lambda}{\lambda\gamma n}\frac{np}{mq}} + 2\max\{\frac{n}{m}, \frac{p}{q}\}} \tag{5.4.9}$$

in Algorithm 3. For each $t \geq 0$, Algorithm 3 guarantees

$$\begin{aligned}
&\mathbb{E}\left[P(x^{(t)}) - D(y^{(t)})\right] \\
\leq\ &\left(1 - \frac{1}{2\sqrt{\frac{\Lambda}{\lambda\gamma n}\frac{np}{mq}} + 2\max\{\frac{n}{m}, \frac{p}{q}\}}\right)^t \times \left\{\frac{1}{\min\left\{\frac{p}{q}, \frac{n}{m}\right\}} + \frac{\max\left\{\frac{\|A\|^2}{n\gamma}, \frac{\|A\|^2}{\lambda n^2}\right\}}{\min\left\{\frac{\lambda p}{q}, \frac{\gamma}{m}\right\}}\right\} \times \\
&\left[\left(\frac{p}{2q\tau} + \frac{p\lambda}{2q}\right)\|x^{(0)} - x^\star\|^2 + \left(\frac{n}{2m\sigma} + \frac{\gamma}{2m}\right)\|y^{(0)} - y^\star\|^2 + \max\left\{\frac{p}{q}, \frac{n}{m}\right\}\left(P(x^{(0)}) - D(y^{(0)})\right)\right].
\end{aligned}$$

According to Theorem 5.4.1 and 5.4.2, in order to obtain a pair of primal and dual solutions with an expected ϵ distance to $(x^\star, y^\star)$, i.e., $\mathbb{E}[\|x^{(t)} - x^\star\|^2] \leq \epsilon$ and $\mathbb{E}[\|y^{(t)} - y^\star\|^2] \leq \epsilon$, or with an expected ϵ objective gap, Algorithm 3 needs

$$t = O\left(\left(\max\left\{\frac{p}{q}, \frac{n}{m}\right\} + \sqrt{\frac{\Lambda_{q,m}}{n\lambda\gamma}\frac{pn}{qm}}\right)\log\left(\frac{1}{\epsilon}\right)\right)$$

iterations when $\Lambda = \Lambda_{q,m}$. This iteration complexity is interesting since it matches the optimal $O((n + \sqrt{n\kappa})\log\left(\frac{1}{\epsilon}\right))$ iteration complexity of dual coordinate methods such as SPDC [220] and others [103, 168] when $(q, m) = (p, 1)$, and also matches the optimal $O((p + p\sqrt{\frac{\Lambda_{1,n}}{n\lambda\gamma}})\log\left(\frac{1}{\epsilon}\right))$ iteration complexity of primal coordinate methods [103, 106] when $(q, m) = (1, n)$.

To efficiently implement Algorithm 3, we just need to maintain and efficiently update either $A\bar{x}^{(t)}$ or $A^T\bar{y}^{(t)}$, depending on whether $\frac{n}{m}$ or $\frac{p}{q}$ is larger. If $\frac{n}{m} \geq \frac{p}{q}$, we should maintain $A^T\bar{y}^{(t)}$ during the algorithm, which is used in (5.4.3) and can be updated in $O(mp)$ time. We will then directly compute $\langle A_i, \bar{x}^{(t)}\rangle$ for $i \in I$ in (5.4.1) in $O(mp)$ time. In fact, this is how SPDC

is implemented in [220] where $q = p$. On the other hand, if $\frac{n}{m} \leq \frac{p}{q}$, it is more efficient to maintain $A\bar{x}^{(t)}$ and update it in $O(qn)$ time and compute $\langle A^j, \bar{y}^{(t+1)} \rangle$ for $j \in J$ in (5.4.3) in $O(qn)$ time. Hence, the overall complexity for DSPDC to find an ϵ-optimal solution is $O((np + \sqrt{\frac{n\Lambda_{q,m}}{\lambda\gamma}}\frac{p^2}{q})\log\left(\frac{1}{\epsilon}\right))$ when $\frac{n}{m} \geq \frac{p}{q}$ and $O((np + \sqrt{\frac{n\Lambda_{q,m}}{\lambda\gamma}}\frac{np}{m})\log\left(\frac{1}{\epsilon}\right))$ when $\frac{n}{m} \leq \frac{p}{q}$. Since the overall complexity of SPDC is $O\left((np + \sqrt{\kappa nm}p)\log\left(\frac{1}{\epsilon}\right)\right)$ when $\frac{n}{m} \geq \frac{p}{q}$, DSPDC method is not more efficient for general data matrix. However, in the next section, we show that DSPDC has an efficient implementation for factorized data matrix which leads to a lower overall complexity than SPDC with the same implementation.

5.4.2 Efficient Implementation for Factorized Data Matrix

In this section, we assume that the data matrix A in (5.1.5) has a factorized structure $A = UV$ where $U \in \mathbb{R}^{n\times d}$ and $V \in \mathbb{R}^{d\times p}$ with $d < \min\{n, p\}$. Such a matrix A is often obtained as a low-rank or denoised approximation of raw data matrix. Recently, there emerges a surge of interests of using factorized data to alleviate the computational cost for big data. For example, Pham and Ghaoui [143] proposed to use a low-rank approximation $X \approx UV = A$ for data matrix X to solve multiple instances of lasso problems. For solving big data kernel learning problems, the Nyström methods, that approximates a $n \times n$ kernel matrix K by $US^{\dagger}U^{\top}$ with $U \in \mathbb{R}^{n\times d}$, $S \in \mathbb{R}^{d\times d}$ and $d < n$, has become a popular method [205]. Moreover, recent advances on fast randomized algorithms [57] for finding a low-rank approximation of a matrix render the proposed coordinate optimization algorithm more attractive for tackling factorized big data problems.

The factorized A also appears often in the problem of sparse recovery from the randomized feature reduction or randomized instance reduction of (5.1.1). The sparse recovery problem from randomized feature reduction can be also formulated into (5.1.5) as

$$\min_{x\in\mathbb{R}^p}\max_{y\in\mathbb{R}^n}\left\{\frac{\lambda_2}{2}\|x\|_2^2 + \lambda_1\|x\|_1 + \frac{1}{n}y^T XG^T Gx - \frac{1}{n}\sum_{i=1}^n \phi_i^*(y_i)\right\} \tag{5.4.10}$$

where X is the original $n \times p$ raw data matrix, G is a $d \times p$ random measurement matrix with $d < p$, and the actual data matrix for (5.1.5) is $A = XG^T G$ with $U = XG^T$ and $V = G$. This approximation approach has been employed to reduce the computational cost of solving underconstrained least-squares problem [112, 186]. Similarly, the randomized instance reduction [45, 186] can be applied by replacing $XG^T G$ in (5.4.10) with $G^T GX$, where G is a $d \times n$ random measurement matrix with $d < n$, and the data matrix $A = G^T GX$ with $U = G^T$ and $V = GX$.

To solve (5.1.5) with $A = UV$, we implement DSPDC by maintaining the vectors $\bar{u}^{(t)} = U^T\bar{y}^{(t)}$ and $\bar{v}^{(t)} = V\bar{x}^{(t)}$ and updating them in $O(dm)$ and $O(dq)$ time, respectively, in each iteration. Then, we can obtain $\langle A_i, \bar{x}^{(t)} \rangle$ in (5.4.1) in $O(dm)$ time by evaluating $\langle U_i, \bar{v}^{(t)} \rangle$ for each $i \in I$, where U_i is the ith row of U. Similarly, we can obtain $\langle A_j, \bar{y}^{(t+1)} \rangle$ in (5.4.3) in $O(dq)$ time by taking $\langle V^j, \bar{v}^{(t)} \rangle$ for each $j \in J$, where V^j is the jth column of V. This leads to an efficient implementation of DSPDC described as in Algorithm 4 whose per-iteration cost is $O(dm + dq)$, lower than the $O(mp)$ or $O(qn)$ cost when A is not factorized.

To make a clear comparison between DSPDC and other methods when applied to factorized data, in Table 5.1, we summarize their numbers of iterations and per-iteration costs (when $A =$

Algorithm 4 Efficient Implementation of Algorithm 3 for Factorized Data

Input: $x^{(-1)} = x^{(0)} = \bar{x}^{(0)} \in \mathbb{R}^p$, $y^{(-1)} = y^{(0)} = \bar{y}^{(0)} \in \mathbb{R}^n$, and parameters (θ, τ, σ)

Initialize: $u^{(0)} = U^T y^{(0)}$, $v^{(0)} = V x^{(0)}$,$\bar{u}^{(0)} = U^T \bar{y}^{(0)}$, $\bar{v}^{(0)} = V \bar{x}^{(0)}$

Iterate:

For $t = 0, 1, 2, \ldots, T-1$

Uniformly and randomly choose $I \subset [n]$ and $J \subset [p]$ of sizes m and q, respectively.

$$y_i^{(t+1)} = \begin{cases} \operatorname{argmax}_{\beta \in \mathbb{R}} \left\{ \frac{1}{n} \langle U_i, \bar{v}^{(t)} \rangle \beta - \frac{\phi_i^*(\beta)}{n} - \frac{1}{2\sigma} (\beta - y_i^{(t)})^2 \right\} & \text{if } i \in I, \\ y_i^{(t)} & \text{if } i \notin I, \end{cases} \tag{5.4.11}$$

$$u^{(t+1)} = u^{(t)} + U^T (y^{(t+1)} - y^{(t)}), \tag{5.4.12}$$

$$\bar{u}^{(t+1)} = u^{(t)} + \frac{n}{m} U^T (y^{(t+1)} - y^{(t)}), \tag{5.4.13}$$

$$x_j^{(t+1)} = \begin{cases} \operatorname{argmin}_{\alpha \in \mathbb{R}} \left\{ \frac{1}{n} \langle V^j, \bar{u}^{(t+1)} \rangle \alpha + g_j(\alpha) + \frac{1}{2\tau} (\alpha - x_j^{(t)})^2 \right\} & \text{if } j \in J, \\ x_j^{(t)} & \text{if } j \notin J, \end{cases} \tag{5.4.14}$$

$$v^{(t+1)} = v^{(t)} + V (x^{(t+1)} - x^{(t)}), \tag{5.4.15}$$

$$\bar{v}^{(t+1)} = v^{(t)} + (\theta + 1) V (x^{(t+1)} - x^{(t)}). \tag{5.4.16}$$

Output: $x^{(T)}$ and $y^{(T)}$

UV)[5]. For all methods in comparison, we assume A is too large so that only U and V are stored in memory, which is the typical situation when applying random reduction. Moreover, the aforementioned efficient implementation in DSPDC (if applicable) has been also applied to other methods to reduce their per-iteration cost. In Table 5.1, we assume $n \geq p$ and $(q, m) = (1, 1)$ and omit all the big-O notations for simplicity. For ASVRG-SP, we present the complexity of its individual-split version with uniform sampling. According to the last column of Table 5.1, our DSPDC with efficient implementation has the lowest overall complexity among these methods.

5.5 Extension with Block Coordinate Updates

With block-wise sampling and updates, DSPDC can be easily generalized and applied to the bilinear saddle-point problem (5.1.5) with a block-decomposable structure and a similar linear convergence rate can be obtained. Although this is a straightforward extension, it is worth showing that, when the proximal mapping on each block is computationally expensive, DSPDC can achieve a lower complexity than other coordinate methods. In this section, we first extend DSPDC to its block coordinate update version, and then identify the scenarios where such an extension has a lower overall complexity than other methods.

[5]For SVRG and ASVRG-SP, we present their numbers of outer and inner iterations and per-iteration costs separately.

Algorithm	Num. of Iter. ($\times \log(\frac{1}{\epsilon})$)	Per-Iter. Cost	Overall Compl. ($\times \log(\frac{1}{\epsilon})$)
DSPDC	$n + p\sqrt{\frac{\Lambda_{1,1} n}{\lambda\gamma}}$	d	$nd + pd\sqrt{\frac{\Lambda_{1,1} n}{\lambda\gamma}}$
SPDC ASDCA APCG RPDG	$n + \sqrt{\frac{\Lambda_{p,1} n}{\lambda\gamma}}$	pd	$npd + pd\sqrt{\frac{\Lambda_{p,1} n}{\lambda\gamma}}$
SDCA SAGA	$n + \frac{\Lambda_{p,1}}{\lambda\gamma}$	pd	$npd + pd\frac{\Lambda_{p,1}}{\lambda\gamma}$
SVRG	Outer: 1 Inner: $\frac{\Lambda_{p,1}}{\lambda\gamma}$	nd pd	$nd + pd\frac{\Lambda_{p,1}}{\lambda\gamma}$
ASVRG-SP	Outer: $\sqrt{\frac{p\max\{\Lambda_{p,1},\Lambda_{1,n}\}}{\lambda\gamma}} + 1$ Inner: $n\sqrt{\frac{p\max\{\Lambda_{p,1},\Lambda_{1,n}\}}{\lambda\gamma}}$	nd d	$nd + nd\sqrt{\frac{p\max\{\Lambda_{p,1},\Lambda_{1,n}\}}{\lambda\gamma}}$

Table 5.1: The overall complexity of finding an ϵ-optimal solution when $A = UV$, $n \geq p$ and U and V (but not A) are stored in memory. We choose $(q, m) = (1, 1)$ in DSPDC.

5.5.1 Algorithm and Convergence Properties

We partition the space $\mathbb{R}^{\bar{p}}$ into p subspaces as $\mathbb{R}^{\bar{p}} = \mathbb{R}^{q_1} \times \mathbb{R}^{q_2} \times \cdots \times \mathbb{R}^{q_p}$ such that $\sum_{j=1}^{p} q_j = \bar{p}$ and partition the space $\mathbb{R}^{\bar{n}}$ into n subspaces as $\mathbb{R}^{\bar{n}} = \mathbb{R}^{m_1} \times \mathbb{R}^{m_2} \times \cdots \times \mathbb{R}^{m_n}$ such that $\sum_{i=1}^{n} m_i = \bar{n}$. With a little abuse of notation, we represent the corresponding partitions of $\mathbf{x} \in \mathbb{R}^{\bar{p}}$ and $\mathbf{y} \in \mathbb{R}^{\bar{n}}$ as $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_p)$ with $\mathbf{x}_j \in \mathbb{R}^{q_j}$ for $j = 1, \ldots, p$ and $\mathbf{y} = (\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_n)$ with $\mathbf{y}_i \in \mathbb{R}^{m_i}$ for $i = 1, \ldots, n$, respectively.

We consider the following bilinear saddle-point problem

$$\min_{\mathbf{x}\in\mathbb{R}^{\bar{p}}} \max_{\mathbf{y}\in\mathbb{R}^{\bar{n}}} \left\{ \sum_{j=1}^{p} g_j(\mathbf{x}_j) + \frac{1}{n}\mathbf{y}^T\mathbf{A}\mathbf{x} - \frac{1}{n}\sum_{i=1}^{n} \phi_i^*(\mathbf{y}_i) \right\}, \tag{5.5.1}$$

where $g_j : \mathbb{R}^{q_j} \to \mathbb{R}$ and $\phi_i^* : \mathbb{R}^{m_i} \to \mathbb{R}$ are functions of $\mathbf{x}_j$ and $\mathbf{y}_i$, respectively. Moreover, we assume g_j and ϕ_i^* are strongly convex with strong convexity parameters of $\lambda > 0$ and $\gamma > 0$, respectively. Due to the partitions on $\mathbf{x} \in \mathbb{R}^{\bar{p}}$ and $\mathbf{y} \in \mathbb{R}^{\bar{n}}$, we partition the matrix $\mathbf{A}$ into blocks accordingly so that

$$\mathbf{y}^T\mathbf{A}\mathbf{x} = \sum_{j=1}^{p}\sum_{i=1}^{n} \mathbf{y}_i^T\mathbf{A}_i^j\mathbf{x}_j,$$

where $\mathbf{A}_i^j \in \mathbb{R}^{m_i \times q_j}$ is the block of $\mathbf{A}$ corresponding to $\mathbf{x}_j$ and $\mathbf{y}_i$.

It is easy to see that the problem (5.1.5) is a special case of (5.5.1) when $q_j = m_i = 1$ for $j = 1, \ldots, p$ and $i = 1, \ldots, n$, $\bar{p} = p$ and $\bar{n} = n$. The scale constant defined in (5.1.8) can be similarly generalized as

$$\mathbf{\Lambda}_{q,m} \equiv \max_{I\subset[n], J\subset[p], |I|=m, |J|=q} \|\mathbf{A}_I^J\|_2^2, \tag{5.5.2}$$

where $\mathbf{A}_I^J$ is sub-matrix of $\mathbf{A}$ consisting of each block $\mathbf{A}_i^j$ with $i \in I$ and $j \in J$.

Let $\mathbf{A}_i = (\mathbf{A}_i^1, \cdots, \mathbf{A}_i^p)$ and $\mathbf{A}^j = ((\mathbf{A}_1^j)^T, \cdots, (\mathbf{A}_n^j)^T)^T$. Given these correspondings between (5.1.5) and (5.5.1), DSPDC can be easily extended for solving (5.5.1) by replacing (5.4.1) and (5.4.3) with

$$\mathbf{y}_i^{(t+1)} = \begin{cases} \operatorname{argmax}_{\beta \in \mathbb{R}^{m_i}} \left\{ \frac{1}{n} \beta^T \mathbf{A}_i \bar{\mathbf{x}}^{(t)} - \frac{\phi_i^*(\beta)}{n} - \frac{1}{2\sigma} \|\beta - \mathbf{y}_i^{(t)}\|^2 \right\} & \text{if } i \in I, \\ \mathbf{y}_i^{(t)} & \text{if } i \notin I, \end{cases} \tag{5.5.3}$$

$$\mathbf{x}_j^{(t+1)} = \begin{cases} \operatorname{argmin}_{\alpha \in \mathbb{R}^{q_j}} \left\{ \frac{1}{n} \alpha^T (\mathbf{A}^j)^T \bar{\mathbf{y}}^{(t+1)} + g_j(\alpha) + \frac{1}{2\tau} \|\alpha - \mathbf{x}_i^{(t)}\|^2 \right\} & \text{if } j \in J, \\ \mathbf{x}_j^{(t)} & \text{if } j \notin J, \end{cases} \tag{5.5.4}$$

respectively, and $\bar{\mathbf{y}}^{(t)}$ and $\bar{\mathbf{x}}^{(t)}$ are updated in the same way as (5.4.2) and (5.4.4).

For this extension, the convergence results similar to Theorem 5.4.1 and Theorem 5.4.2 can be easily derived with almost the same proof. We skip the proofs but directly state the results. To find a pair of primal-dual solutions for (5.5.1) which either has an ϵ-distance to the optimal solution or has an ϵ-primal-dual objective gap, the number of iterations Algorithm 3 (with by (5.4.1) and (5.4.3) replaced by (5.5.3) and (5.5.4)) needs is

$$t = O\left(\left(\max\left\{ \frac{n}{m}, \frac{p}{q} \right\} + \sqrt{\frac{\mathbf{\Lambda}_{q,m}}{\lambda \gamma n} \frac{np}{mq}} \right) \log\left(\frac{1}{\epsilon} \right) \right).$$

5.5.2 Application 1: Matrix Risk Minimization

In this section, we study the theoretical performance of DSPDC method when the block updating step (5.5.3) or (5.5.4) has a high computational cost due to eigenvalue decomposition. Let $\mathbb{S}_+^d$ be the set of $d \times d$ positive semi-definite matrices. The problem we consider is a general multiple-matrix risk minimization which is formulated as

$$\min_{X_j \in \mathbb{S}_+^d, j=1,\ldots,p} \left\{ \frac{1}{n} \sum_{i=1}^n \phi_i \left(\sum_{j=1}^p \langle \mathbf{D}_i^j, X_j \rangle \right) + \frac{\lambda}{2} \sum_{j=1}^p \|X_j\|_F^2 \right\}, \tag{5.5.5}$$

where $\mathbf{D}_i^j$ is a $d \times d$ data matrix, ϕ_i is $(1/\gamma)$-smooth convex loss function applied to the linear prediction $\sum_{j=1}^p \langle \mathbf{D}_i^j, X_j \rangle$ and λ is a regularization parameter. The associated saddle-point formulation of (5.5.5) is

$$\min_{X_j \in \mathbb{S}_+^d, j=1,\ldots,p} \max_{y \in \mathbb{R}^n} \left\{ \frac{\lambda}{2} \sum_{j=1}^p \|X_j\|_F^2 + \frac{1}{n} \sum_{i=1}^n \sum_{j=1}^p y_i \langle \mathbf{D}_i^j, X_j \rangle - \frac{1}{n} \sum_{i=1}^n \phi_i^*(y_i) \right\}, \tag{5.5.6}$$

which is a special case of (5.5.1) where $q_j = d^2$ and $m_i = 1$, $\mathbf{x}_j \in \mathbb{R}^{d^2}$ and $\mathbf{A}_i^j \in \mathbb{R}^{1 \times d^2}$ are the vectorization of the matrices X_j and $\mathbf{D}_i^j$ respectively, and $g_j(X_j) = \frac{\lambda}{2} \|X_j\|_F^2$ if $X_j \in \mathbb{S}_+^d$ and $g_j(X_j) = +\infty$ if $X_j \notin \mathbb{S}_+^d$. The applications of this model include matrix trace regression [175] and distance metric learning [138, 191, 192].

In Table 5.2, we compare DSPDC with various methods on the numbers of iterations and per-iteration costs when applied to problem (5.5.6). We assume $n \geq \max\{p, d\}$ and $(q, m) =$

Algorithm	Num. of Iter. ($\times \log(\frac{1}{\epsilon})$)	Per-Iter. Cost	Overall Compl. ($\times \log(\frac{1}{\epsilon})$)
DSPDC	$n + p\sqrt{\frac{\mathbf{\Lambda}_{1,1} n}{\lambda\gamma}}$	$pd^2 + d^3$	$(d^2 p + d^3)(n + p\sqrt{\frac{\mathbf{\Lambda}_{1,1} n}{\lambda\gamma}})$
SPDC ASDCA APCG RPDG	$n + \sqrt{\frac{\mathbf{\Lambda}_{p,1} n}{\lambda\gamma}}$	pd^3	$npd^3 + pd^3\sqrt{\frac{\mathbf{\Lambda}_{p,1} n}{\lambda\gamma}}$
SDCA SAGA	$n + \frac{\mathbf{\Lambda}_{p,1}}{\lambda\gamma}$	pd^3	$npd^3 + pd^3\frac{\mathbf{\Lambda}_{p,1}}{\lambda\gamma}$
SVRG	Outer: 1 Inner: $\frac{\mathbf{\Lambda}_{p,1}}{\lambda\gamma}$	pnd^2 pd^3	$npd^2 + pd^3\frac{\mathbf{\Lambda}_{p,1}}{\lambda\gamma}$
ASVRG-SP	Outer: $\sqrt{\frac{d\max\{\mathbf{\Lambda}_{p,1}, \mathbf{\Lambda}_{1,n}\}}{\lambda\gamma}} + 1$ Inner: $np\sqrt{\frac{\max\{\mathbf{\Lambda}_{p,1}, \mathbf{\Lambda}_{1,n}\}}{d\lambda\gamma}}$	pnd^2 d^3	$npd^2 + pnd^2\sqrt{\frac{d\max\{\mathbf{\Lambda}_{p,1}, \mathbf{\Lambda}_{1,n}\}}{\lambda\gamma}}$

Table 5.2: The overall complexity of finding an ϵ-optimal solution for (5.5.6) when $n \geq p$.

$(1, 1)$ and omit all the big-O notations for simplicity. For ASVRG-SP, we present the complexity of its individual-split version with uniform sampling. When applied to (5.5.6) with $(q, m) = (1, 1)$, DSPDC requires solving (5.5.4) in each iteration which involves the eigenvalue decomposition of one $d \times d$ matrix with complexity of $O(d^3)$. To efficiently implement DSPDC, we need to maintain and efficiently update either $\mathbf{A}\bar{\mathbf{x}}^{(t)}$ or $\mathbf{A}^T\bar{\mathbf{y}}^{(t)}$ with complexity of $O(d^2 \min\{n, p\})$. When $p \leq n$, the per-iteration cost of DSPDC in this case is therefore $O(d^3 + pd^2)$ so that the overall complexity for DSPDC to find an ϵ-optimal solution of (5.5.6) is $O((d^3 + pd^2)(n + \sqrt{\frac{n\mathbf{\Lambda}_{1,1}}{\lambda\gamma}}p)\log(1/\epsilon))$. On the contrary, SPDC, ASDCA, APCG and RPDG need to solve p eigenvalue decompositions per iteration so that its the overall complexity is $O(pd^3(n + \sqrt{\frac{n\mathbf{\Lambda}_{p,1}}{\lambda\gamma}})\log(1/\epsilon))$ which is higher than that of DSPDC when $\sqrt{\mathbf{\Lambda}_{1,1}}p \leq \sqrt{\mathbf{\Lambda}_{p,1}}d$. Without this condition, according to the last column of Table 5.2, DSPDC still has a lower overall complexity than SDCA, SAGA, SVRG and ASVRG-SP.

5.5.3 Application 2: Multi-task Large Margin Nearest Neighbor Problem

In this section, we show that DSPDC can be applied to the Multi-task Large Margin Nearest Neighbor (MT-LMNN) problem [138]. The key is to appropriately reduce the original form to the matrix risk minimization (5.5.6).

Problem Reformulation To make this chapter self-contained, we include the introduction of MT-LMNN here. Interested readers can find more background in [138]. Suppose there are $p > 1$ tasks, each being a multi-class classification problem. For example, in our empirical study (Section 5.6.3), we have $p = 100$ tasks, each being a 10-class image classification problem. MT-LMNN aims to learn one Mahalanobis distance metric (defined as a positive semi-definite matrix) for each task, so there are totally p metric matrices to be learned. With those metrics, the label of a testing point in task j is determined by the majority vote of its ℓ-nearest neighbors defined by the j-th metric. We further assume that the tasks are correlated that all the metrics

share a common component, in addition to their own matrix. The original formulation of MT-LMNN is the following:

$$\begin{aligned}\min_{X_j\in\mathbb{S}^d_+,j=0,1,\ldots,p,\xi\in\mathbb{R}^n} \quad & \frac{\lambda_0}{2}\|X_0-I\|_F^2+\sum_{j=1}^p\frac{\lambda_j}{2}\|X_j\|_F^2+\frac{1}{n}\sum_{j=1}^p\sum_{(u,v)\in\mathcal{N}_j}d_j^2(z_u,z_v)+\frac{1}{n}\sum_{j=1}^p\sum_{(u,v,w)\in\mathcal{S}_j}\xi_{uvw}\\ \text{s.t.} \quad & d_j^2(z_u,z_w)-d_j^2(z_u,z_v)\geq 1-\xi_{uvw},\quad \forall j\in[p],\ \forall(u,v,w)\in\mathcal{S}_j \\ & \xi_{uvw}\geq 0,\quad \forall(u,v,w)\in\mathcal{S}_j.\end{aligned} \tag{5.5.7}$$

Now we interpret the notations above. We let z denote a training data point indexed by subscript u, v, w etc, $X_j \in \mathbb{S}^d_+$ be the metric matrix for task $j = 1, 2, ..., p$ and X_0 the common component shared by all the tasks to reflect the correlations among them. Let $\mathcal{N}_j$ be the set of every ordered pair (u, v) for task j such that z_v is among the ℓ closest points of z_u that has the same label as z_u, and $\mathcal{S}_j$ be the set consisting of the triples (u, v, w) such that $(u, v) \in \mathcal{N}_j$ and z_w is the closest point to z_u that has a different label. The aforementioned closeness can be measured in Euclidean distance or other appropriate methods in the original feature space. We use $n := \sum_{j=1}^p |\mathcal{N}_j| = \sum_{j=1}^p |\mathcal{S}_j|$ to denote the total number of constraints in (5.5.7) excluding the non-negativity constraints. Let $\mathbf{Z}_{j,uv} := (z_u - z_v)(z_u - z_v)^\top$ for all $(u, v) \in \mathcal{N}_j$ and $\mathbf{Z}_{j,uvw} := \mathbf{Z}_{j,uw} - \mathbf{Z}_{j,uv}$ for all $(u, v, w) \in \mathcal{S}_j$ so that the distance $d_j(z_u, z_v)$ in task j is defined as

$$d_j(z_u,z_v)=\sqrt{(z_u-z_v)^\top(X_j+X_0)(z_u-z_v)}=\sqrt{\langle\mathbf{Z}_{j,uv},X_j+X_0\rangle},$$

Note that the metric matrix for task j is $X_j + X_0$, the sum of the individual matrix X_j and the shared component X_0 among all the tasks. We can see that in each task, the goal of formulation (5.5.7) is essentially to minimize the distances of points with the same label (the objective) while enforcing the points with different labels to stay away from each other (the constraints). The slack variables ξ_{uvw} allow for soft constraints in the problem. The regularization term $\lambda_j\|X_j\|_F^2, \forall j \in [p]$ controls the magnitude of X_j and $\lambda_0\|X_0 - I\|_F^2$ tunes how close X_0 to the identity I.

Following the same convention of support vector machine, we can transform the problem (5.5.7) to an unconstrained form:

$$\begin{aligned}\min_{X_0,\ldots,X_p\in\mathbb{S}^d_+} \quad & \frac{\lambda_0}{2}\|X_0-I\|_F^2+\sum_{j=1}^p\frac{\lambda_j}{2}\|X_j\|_F^2+\frac{1}{n}\sum_{j=1}^p\sum_{(u,v)\in\mathcal{N}_j}\langle\mathbf{Z}_{j,uv},X_j+X_0\rangle \\ & +\frac{1}{n}\sum_{j=1}^p\sum_{(u,v,w)\in S_j}\phi(\langle\mathbf{Z}_{j,uvw},X_j+X_0\rangle).\end{aligned} \tag{5.5.8}$$

where $\phi(\cdot)$ is the hinge loss and we adopt its smoothed version (5.1.3) with $b = 1$.

By introducing the dual variable y we can obtain the following equivalent saddle-point formulation of (5.5.8):

$$\begin{aligned}\min_{X_0,\ldots,X_p\in\mathbb{S}^d_+}\max_{y\in\mathbb{R}^n} \quad & \frac{\lambda_0}{2}\|X_0-I\|_F^2+\sum_{j=1}^p\frac{\lambda_j}{2}\|X_j\|_F^2+\frac{1}{n}\sum_{j=1}^p\sum_{(u,v)\in\mathcal{N}_j}\langle\mathbf{Z}_{j,uv},X_j+X_0\rangle \\ & +\frac{1}{n}\sum_{j=1}^p\sum_{(u,v,w)\in\mathcal{S}_j}y_{j,uvw}\langle\mathbf{Z}_{j,uvw},X_j+X_0\rangle-\frac{1}{n}\sum_{j=1}^p\sum_{(u,v,w)\in\mathcal{S}_j}\phi^*(y_{j,uvw}).\end{aligned} \tag{5.5.9}$$

Algorithm 5 DSPDC Customized for MT-LMNN

Input: $X^{(-1)} = X^0 = \bar{X}^0 \in \mathbb{R}^{d\times d}$, $y^{(-1)} = y^0 = \bar{y}^0 \in \mathbb{R}^n$, step sizes τ, σ, parameter θ, total iteration S, sample sizes $1 \le m \le n$ and $1 \le q \le p+1$.

Output: X^S and y^S;

$\mathbf{W}_j^0 = \sum_{i:i\in[n],\mathcal{T}(i)=j} \bar{y}_i^0 \mathbf{Z}_i$ for $j = 1, 2, ..., p$ and $\mathbf{W}_0^0 = \sum_{j=1}^{p} \mathbf{W}_j^0 = \sum_{i=1}^{n} \bar{y}_i^0 \mathbf{Z}_i$.

$\mathbf{B}_j^0 = 0_{d\times d}$ for $j = 1, 2, ..., p$.

For $t = 0, 1, 2, \ldots, S-1$:

Randomly choose $\mathcal{I}_t \subset \{1, 2, ..., n\}$ and $\mathcal{J}_t \subset \{0, 1, ..., p\}$ with $|\mathcal{I}_t| = m$ and $|\mathcal{J}_t| = q$. Perform the following updates:

$$y_i^{t+1} = \begin{cases} \underset{\beta\in\mathbb{R}}{\operatorname{argmax}} \left\{ \frac{\beta}{n} \left\langle \mathbf{Z}_i, \bar{X}_{\mathcal{T}(i)}^t + \bar{X}_0^t \right\rangle - \frac{1}{n}\phi^*(\beta) - \frac{1}{2\sigma}(\beta - y_i^t)^2 \right\} & \text{if } i \in \mathcal{I}_t, \\ y_i^t & \text{if } i \notin \mathcal{I}_t \end{cases} \tag{5.5.11}$$

$$\mathbf{B}_j^{t+1} = \sum_{i:i\in\mathcal{I}_t,\mathcal{T}(i)=j} (y_i^{t+1} - y_i^t)\mathbf{Z}_i, \quad j = 1, 2, ..., p \tag{5.5.12}$$

$$\mathbf{W}_j^{t+1} = \begin{cases} \mathbf{W}_j^t + \frac{n}{m}\mathbf{B}_j^{t+1} + \frac{m-n}{m}\mathbf{B}_j^t & \text{if } j \neq 0, \\ \sum_{l=1}^{T} \mathbf{W}_l^{t+1} & \text{if } j = 0. \end{cases} \tag{5.5.13}$$

$$X_j^{t+1} = \begin{cases} \underset{Q\in\mathbb{S}_+^d}{\operatorname{argmin}} \left\{ \frac{1}{n}\langle \mathbf{W}_j^{t+1}, Q\rangle + g_j(Q) + \frac{1}{2\tau}\|Q - X_j^t\|_F^2 \right\} & \text{if } j \in \mathcal{J}_t, \\ X_j^t & \text{if } j \notin \mathcal{J}_t \end{cases} \tag{5.5.14}$$

$$\bar{X}_j^{t+1} = X_j^t + (\theta+1)(X_j^{t+1} - X_j^t), \ \forall j = 0, 1, ..., p. \tag{5.5.15}$$

Here, each dual variable $y_{j,uvw}$ corresponds to the matrix $\mathbf{Z}_{j,uvw}$ and the constraint $d_j^2(z_u, z_w) - d_j^2(z_u, z_v) \ge 1 - \xi_{uvw}$ in (5.5.7) for all $j \in [p]$ and $(u, v, w) \in \mathcal{S}_j$. We stack all the dual variables $y_{j,uvw}$ into a single column vector $y \in \mathbb{R}^n$ and y_s represents the sth coordinate of y. Let $\mathcal{T}(s)$ and $\mathbf{Z}_s$ represent the task and the outer product y_s corresponds to, namely, $\mathcal{T}(s) = j$ and $\mathbf{Z}_s := \mathbf{Z}_{j,uw} - \mathbf{Z}_{j,uv}$ if the new index s corresponds to the original index (j, uvw). Then we have the following more compact formulation:

$$\min_{X_0,...,X_p\in\mathbb{S}_+^d} \max_{y\in\mathbb{R}^n} \sum_{j=0}^{p} g_j(X_j) + \frac{1}{n}\sum_{i=1}^{n} y_i\langle \mathbf{Z}_i, X_{\mathcal{T}(i)} + X_0\rangle - \frac{1}{n}\sum_{i=1}^{n}\phi^*(y_i), \tag{5.5.10}$$

where $g_j(X_j) := \frac{\lambda_j}{2}\|X_j\|_F^2 + \frac{1}{n}\langle \mathbf{C}_j, X_j\rangle$, with $\mathbf{C}_j = \sum_{(u,v)\in\mathcal{N}_j} \mathbf{Z}_{j,uv}$ for $j \in [p]$, and $g_0(X_0) := \frac{\lambda_0}{2}\|X_0 - I\|_F^2 + \frac{1}{n}\langle \mathbf{C}_0, X_0\rangle$ with $\mathbf{C}_0 = \sum_{j=1}^{p} \mathbf{C}_j$. Now, we have reduced the MT-LMNN problem to the form of (5.5.6) and the customized method is shown in Algorithm 5. The convergence properties similar to Theorem 5.4.1 and 5.4.2 immediately follow.

5.6 Numerical Experiments

In this section, we conduct numerical experiments to compare the DSPDC method with two other popular stochastic coordinate methods, SPDC [220] and SDCA [166] on three scenarios. The first two are empirical risk minimizations, with one applied on factorized data (see Section 5.4.2) and the other using matrices as decision variables (see Section 5.5.2), respectively. Those experiments are run on somewhat synthetic data and serve as the first step of sanity check for the convergence speed. The third is a multi-task large margin nearest neighbor metric learning problem (see Section 5.5.3) on a real dataset. In a nutshell, we show that DSPDC outperforms the competitors in terms of running time in all the experiments.

5.6.1 Learning with factorized data

We first consider the binary classification problem with smoothed hinge loss under the sparse recovery setting. Besides, we work on a low-dimensional feature space where random feature reduction is applied. That being said, we are solving the problem (5.4.10) with $\phi_i(z)$ given by (5.1.3).

For the experiments over synthetic data, we first generate a random matrix $X \in \mathbf{R}^{n\times p}$ with X_{ij} following i.i.d. standard normal distribution. We sample a random vector $\beta \in \mathbb{R}^p$ with $\beta_j = 1$ for $j = 1, 2, \ldots, 50$ and $\beta_j = 0$ for $j = 51, 52, \ldots, p$ and use β to randomly generate b_i with the distribution $\Pr(b_i = 1|\beta) = 1/(1 + e^{-X_i^T\beta})$ and $\Pr(b_i = -1|\beta) = 1/(1 + e^{X_i^T\beta})$. To construct factorized data, we generate a random matrix $G \in \mathbf{R}^{d\times p}$ with $d < p$ and G_{ij} following i.i.d. normal distribution $\mathcal{N}(0, 1/d)$. Then, the factorized data $A = UV$ for (5.1.1) is constructed with $U = XG^T$ and $V = G$.

To demonstrate the effectiveness of these three methods under different settings, we choose different values for (n, m, p, q, d) and the regularization parameters (λ_1, λ_2) in (5.4.10). The numerical results are presented in Figure 5.1 with the choices of parameters stated at its bottom. Here, the horizontal axis represents the running time of an algorithm while the vertical axis represents the primal gap in logarithmic scale. According to Figure 5.1, DSPDC is significantly faster than both SPDC and SCDA, under these settings.

We then conduct the comparison of these methods over three real datasets[6]: Covtype ($n = 581012, p = 54$), RCV1 ($n = 20242, p = 47236$), and Real-sim ($n = 72309, p = 20958$). We still consider the sparse recovery problem from feature reduction which is formulated as (5.4.10) with ϕ_i defined as (5.1.3). In all experiments, we choose $d = 20$ to generate the random matrix G and set $\lambda_1 = 10^{-4}$, $\lambda_2 = 10^{-2}$ in (5.4.10). We choose m and q so that n and p can be either dividable by them or has a small division remainder. The numerical performances of the three methods are shown in Figure 5.2. In these three examples, SPDC and DSPDC both outperform SDCA significantly. Compared to SPDC, DSPDC is even better on the first two datasets and has the same efficiency on the third.

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

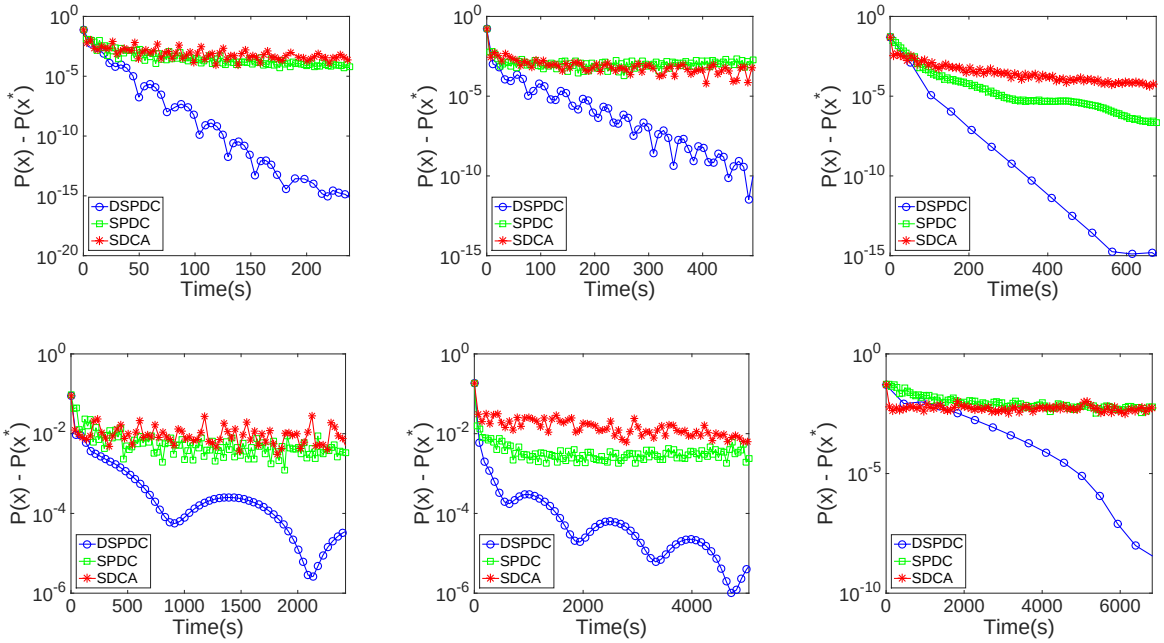

Figure 5.1: For all cases, $m = 1$. First row: $\lambda_1 = 10^{-3}, \lambda_2 = 10^{-2}$; Second row: $\lambda_1 = 10^{-6}, \lambda_2 = 10^{-5}$. First column: $(n, p, q, d) = (5000, 100, 50, 20)$; Second column: $(n, p, q, d) = (10000, 100, 50, 50)$; Third column: $(n, p, q, d) = (10000, 500, 50, 50)$.

5.6.2 Matrix Risk Minimization

Next we study the performance of DSPDC for solving the multiple-matrix risk minimization problem (5.5.5). We choose ϕ_i in (5.5.5) to be (5.1.3) and generate $\mathbf{D}_i^j$ as a $d \times d$ matrix with entry sampled from a standard Gaussian distribution for $i = 1, 2, ..., n$ and $j = 1, 2, ..., p$. Then we generate the true parameter matrix $\bar{X}_j$ as a $d \times d$ identity matrix for $j = 1, 2, ..., p$. Then we use $\bar{X}_j$ and $\mathbf{D}_i^j$ to generate b_i such that $b_i = 1$ if $\frac{1}{(1+\exp\{-\langle \mathbf{D}_i^j, \bar{X}_j \rangle\})} > 0.5$ or $b_i = -1$ otherwise. In this experiment, we set $d = 100$ or 200, $p = 100, n = 100, \lambda = 0.01$.

We compare the performance of DSPDC, SPDC [220] and SDCA [166] with various sampling settings, and the results are shown in Fig 5.3. It can be easily seen that DSPDC converges much faster than both SPDC and SDCA, in terms of running time. The behaviors of these algorithms are due to the fact that, in each iteration, both SPDC and SDCA need to take p eigenvalue decompositions of $d \times d$ matrix while DSPDC only needs q such operations. Since the cost of each eigenvalue decomposition is as expensive as $O(d^3)$, the total computation cost saved by DSPDC is thus significant.

5.6.3 Multi-task Large Margin Nearest Neighbor Problem

Finally, we compare the performance of different algorithms on the MT-LMNN problem (5.5.10). The dataset we are using is Amsterdam library of objects ALOI[7], a collection of 108,000 images

[7] https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multiclass.html#aloi

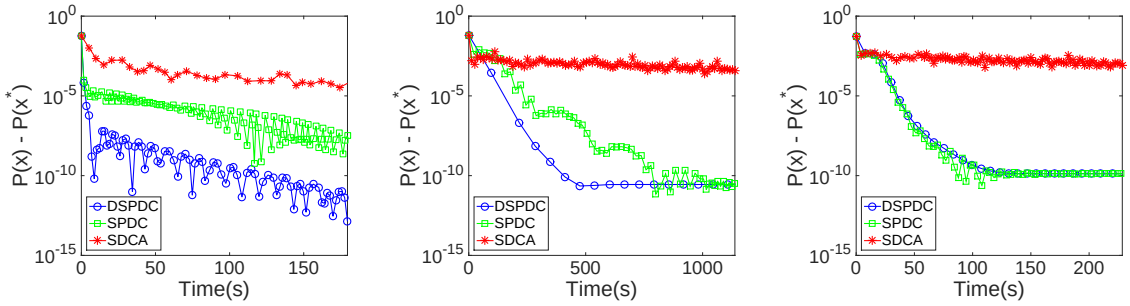

Figure 5.2: Performance on real datasets. Left: Covtype. Middle: RCV1. Right: Real-sim.

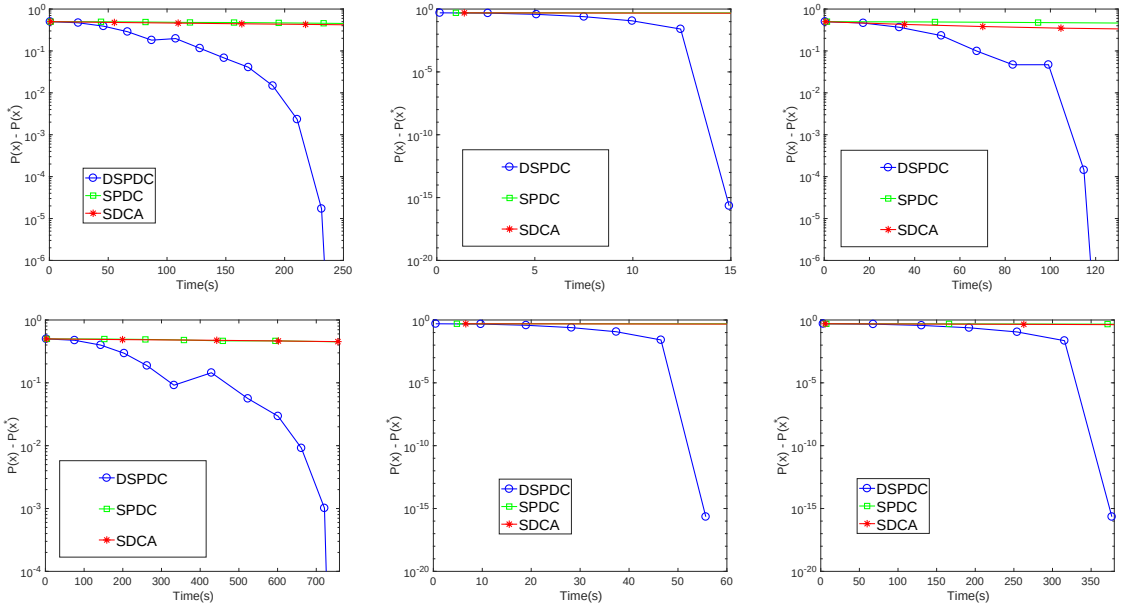

Figure 5.3: Performance on matrix risk minimization. First row: $d = 100$. Second row: $d = 200$. Left: $(m, q) = (5, 50)$. Middle: $(m, q) = (50, 5)$. Right: $(m, q) = (20, 20)$.

for small objects with 1,000 class labels. Each image contains one small object which can be expressed as an extended color histogram of $d = 128$ dimensions. We adopt the approach similar to [138] to generate classification tasks. More specifically, we divide the class labels into 100 pieces, each having 10 labels. In other words, we have 100 metric matrices to learn during training, as well as the one shared by all the tasks. The neighborhood size is $\ell = 3$. For each task, we randomly select 60% of the data for training, resulting in a training set of 63936 instances. Under this setting, the total number of triplet constraints is $n = 1142658$, which is also the number of dual variables. We set $\lambda_0 = 0.01, \lambda_1 = \cdots = \lambda_p = 0.1$.

The comparison is again between DSPDC, SPDC and SDCA under different sampling schemes, which is shown in Figure 5.4. We can observe the similar trends as Figure 5.3. In particular, DSPDC converges much faster to the optimal solution in terms of running time than both SPDC and SDCA, under all the sampling settings. The superiority of DSPDC in terms of running time is again due to the much less eigenvalue decomposition it does per iteration, which is the benefit brought by primal sampling. Indeed, as both SPDC and SDCA need to do full primal coordinate

update, they have to carry out $\frac{p}{q}$ times more eigenvalue decompositions than DSPDC. While all those methods have similar linear convergence rates, the computational cost per iteration dominates the performance.

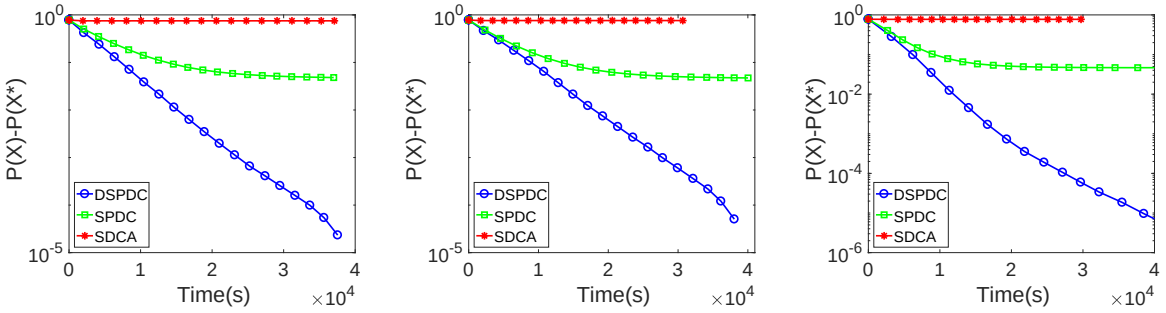

Figure 5.4: The result of different methods on large margin multi-task metric learning problem with ALOI data. There are $p = 100$ tasks and thus $p + 1 = 101$ metric matrices to be learned, each being 128×128. The dual variable (triplet constraint) size is $n = 1142658$. For left to right, the primal and dual sampling sizes of DSPDC are respectively $(q, m) = (20, 2000), (q, m) = (20, 4000)$ and $(q, m) = (40, 8000)$. For SPDC and SDCA, the dual sampling sizes are the same as DSPDC while they conduct full primal coordinate update ($q = 101$).

5.7 Discussion

In this chapter, we propose a doubly stochastic primal dual coordinate (DSPDC) method for bilinear saddle point problem, which captures an important class of regularized empirical risk minimization (ERM) problems in statistical learning. We establish the iteration complexity of DSPDC for finding a pair of primal and dual solutions with ϵ-distance to the optimal solution or with ϵ-objective gap. When applied to ERM with factorized data or matrix variables with costly prox-mapping, such as the multi-task large margin nearest neighbor metric learning problem, our method achieves a lower overall complexity than existing coordinate methods.

Chapter 6

DSCOVR: Randomized Primal-Dual Block Coordinate Algorithms for Asynchronous Distributed Optimization

6.1 Introduction

In the previous chapter, we have shown that empirical risk minimization can be reformulated as a bilinear saddle point problem. In this chapter, we consider how to leverage this reformulation to conduct distributed computation. In particular, we consider distributed optimization problems of the form

$$\underset{w\in\mathbf{R}^d}{\text{minimize}} \quad \frac{1}{m}\sum_{i=1}^{m} f_i(X_i w) + g(w), \tag{6.1.1}$$

where $X_i \in \mathbf{R}^{N_i\times d}$ is the local data stored at the ith machine, $f_i : \mathbf{R}^{N_i} \to \mathbf{R}$ is a convex cost function associated with the linear mapping $X_i w$, and $g(w)$ is a convex regularization function. In addition, we assume that g is separable, i.e., for some integer $n > 0$, we can write

$$g(w) = \sum_{k=1}^{n} g_k(w_k)\,, \tag{6.1.2}$$

where $g_k : \mathbf{R}^{d_k} \to \mathbf{R}$, and $w_k \in \mathbf{R}^{d_k}$ for $k = 1,\ldots,n$ are non-overlapping subvectors of $w \in \mathbf{R}^d$ with $\sum_{k=1}^{n} d_k = d$ (they form a partition of w). Many popular regularization functions in machine learning are separable, for example, $g(w) = (\lambda/2)\|w\|_2^2$ or $g(w) = \lambda\|w\|_1$ for some $\lambda > 0$.

An important special case of (6.1.1) is distributed empirical risk minimization (ERM) of linear predictors. Let $(x_1, y_1), \ldots, (x_N, y_N)$ be N training examples, where each $x_j \in \mathbf{R}^d$ is a feature vector and $y_j \in \mathbf{R}$ is its label. The ERM problem is formulated as

$$\underset{w\in\mathbf{R}^d}{\text{minimize}} \quad \frac{1}{N}\sum_{j=1}^{N} \phi_j\big(x_j^T w\big) + g(w), \tag{6.1.3}$$

where each $\phi_j : \mathbf{R} \to \mathbf{R}$ is a loss function measuring the mismatch between the linear prediction $x_j^T w$ and the label y_j. Popular loss functions in machine learning include, e.g., for regression, the squared loss $\phi_j(t) = (1/2)(t - y_j)^2$, and for classification, the logistic loss $\phi_j(t) = \log(1 + \exp(-y_j t))$ where $y_j \in \{\pm 1\}$. In the distributed optimization setting, the N examples are divided into m subsets, each stored on a different machine. For $i = 1, \ldots, m$, let $\mathcal{I}_i$ denote the subset of $\{1, \ldots, N\}$ stored at machine i and let $N_i = |\mathcal{I}_i|$ (they satisfy $\sum_{i=1}^m N_i = N$). Then the ERM problem (6.1.3) can be written in the form of (6.1.1) by letting X_i consist of x_j^T with $j \in \mathcal{I}_i$ as its rows and defining $f_i : \mathbf{R}^{N_i} \to \mathbf{R}$ as

$$f_i(u_{\mathcal{I}_i}) = \frac{m}{N} \sum_{j \in \mathcal{I}_i} \phi_j(u_j), \tag{6.1.4}$$

where $u_{\mathcal{I}_i} \in \mathbf{R}^{N_i}$ is a subvector of $u \in \mathbf{R}^N$, consisting of u_j with $j \in \mathcal{I}_i$.

The nature of distributed algorithms and their convergence properties largely depend on the model of the communication network that connects the m computing machines. A popular setting in the literature is to model the communication network as a graph, and each node can only communicate (in one step) with their neighbors connected by an edge, either synchronously or asynchronously [e.g., 16, 124]. The convergence rates of distributed algorithms in this setting often depend on characteristics of the graph, such as its diameter and the eigenvalues of the graph Laplacian [e.g. 48, 125, 160, 199]. This is often called the *decentralized* setting.

Another model for the communication network is *centralized*, where all the machines participate synchronous, collective communication, e.g., broadcasting a vector to all m machines, or computing the sum of m vectors, each from a different machine (AllReduce). These collective communication protocols hide the underlying implementation details, which often involve operations on graphs. They are adopted by many popular distributed computing standards and packages, such as MPI [122], MapReduce [38] and Aparche Spark [215], and are widely used in machine learning practice [e.g., 100, 119]. In particular, collective communications are very useful for addressing *data parallelism*, i.e., by allowing different machines to work in parallel to improve the same model $w \in \mathbf{R}^d$ using their local dataset. A disadvantage of collective communications is their synchronization cost: faster machines or machines with less computing tasks have to become idle while waiting for other machines to finish their tasks in order to participate a collective communication.

One effective approach for reducing synchronization cost is to exploit *model parallelism* (here "model" refers to $w \in \mathbf{R}^d$, including all optimization variables). The idea is to allow different machines work in parallel with different versions of the full model or different parts of a common model, with little or no synchronization. The model partitioning approach can be very effective for solving problems with large models (large dimension d). Dedicated *parameter servers* can be set up to store and maintain different subsets of the model parameters, such as the w_k's in (6.1.2), and be responsible for coordinating their updates at different workers [98, 202]. This requires flexible point-to-point communication.

In this paper, we develop a family of randomized algorithms that exploit *simultaneous* data and model parallelism. Correspondingly, we adopt a centralized communication model that support both synchronous collective communication and asynchronous point-to-point communication. In particular, it allows any pair of machines to send/receive a message in a single step, and

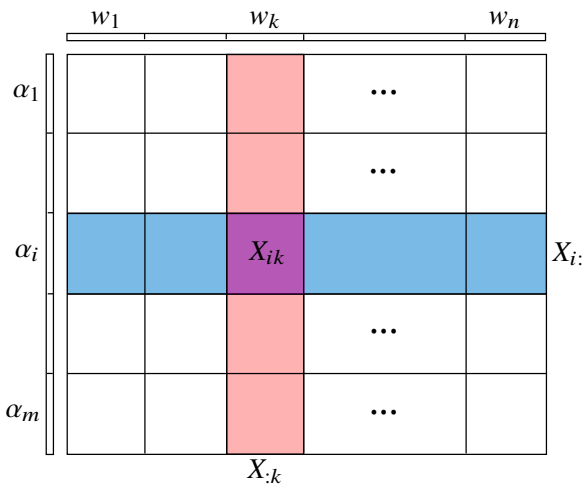

Figure 6.1: Partition of primal variable w, dual variable α, and the data matrix X.

multiple point-to-point communications may happen in parallel in an event-driven, asynchronous manner. Such a communication model is well supported by the MPI standard. To evaluate the performance of distributed algorithms in this setting, we consider the following three measures.

- *Computation complexity:* total amount of computation, measured by the number of passes over all datasets X_i for $i = 1, \ldots, m$, which can happen in parallel on different machines.
- *Communication complexity:* the total amount of communication required, measured by the equivalent number of vectors in $\mathbf{R}^d$ sent or received across all machines.
- *Synchronous communication:* measured by the total number of vectors in $\mathbf{R}^d$ that requires synchronous collective communication involving all m machines. We single it out from the overall communication complexity as a (partial) measure of the synchronization cost.

In Section 6.2, we introduce the framework of our randomized algorithms, **D**oubly **S**tochastic **C**oordinate **O**ptimization with **V**ariance **R**eduction (DSCOVR), and summarize our theoretical results on the three measures achieved by DSCOVR. Compared with other first-order methods for distributed optimization, we show that DSCOVR may require less amount of overall computation and communication, and less or no synchronization. Then we present the details of several DSCOVR variants and their convergence analysis in Sections 6.3-6.6. We discuss the implementation of different DSCOVR algorithms in Section 6.7, and present results of our numerical experiments in Section 6.8.

6.2 The DSCOVR Framework and Main Results

First, we derive a saddle-point formulation of the convex optimization problem (6.1.1). Let f_i^* be the convex conjugate of f_i, i.e., $f_i^*(\alpha_i) = \sup_{u_i \in \mathbf{R}^{N_i}} \left\{\alpha_i^T u_i - f_i(u_i)\right\}$, and define

$$L(w, \alpha) \equiv \frac{1}{m} \sum_{i=1}^{m} \alpha_i^T X_i w - \frac{1}{m} \sum_{i=1}^{m} f_i^*(\alpha_i) + g(w)\,, \tag{6.2.1}$$

where $\alpha = [\alpha_1; \ldots; \alpha_m] \in \mathbf{R}^N$. Since both the f_i's and g are convex, $L(w, \alpha)$ is convex in w and concave in α. We also define a pair of *primal* and *dual* functions:

$$P(w) \;=\; \max_{\alpha \in \mathbf{R}^N} L(w, \alpha) \;=\; \frac{1}{m} \sum_{i=1}^{m} f_i(X_i w) + g(w)\,, \tag{6.2.2}$$

$$D(\alpha) \;=\; \min_{w \in \mathbf{R}^d} L(w, \alpha) \;=\; -\frac{1}{m} \sum_{i=1}^{m} f_i^*(\alpha_i) - g^*\left(-\frac{1}{m} \sum_{i=1}^{m} (X_i)^T \alpha_i\right), \tag{6.2.3}$$

where $P(w)$ is exactly the objective function in (6.1.1)[1] and g^* is the convex conjugate of g. We assume that L has a saddle point $(w^\star, \alpha^\star)$, that is,

$$L(w^\star, \alpha) \leq L(w^\star, \alpha^\star) \leq L(w, \alpha^\star)\,, \qquad \forall (w, \alpha) \in \mathbf{R}^d \times \mathbf{R}^N.$$

In this case, we have $w^\star = \operatorname{argmin}\ P(w)$ and $\alpha^\star = \operatorname{argmin}\ D(\alpha)$, and $P(w^\star) = D(\alpha^\star)$.

The DSCOVR framework is based on solving the convex-concave saddle-point problem

$$\min_{w \in \mathbf{R}^d} \max_{\alpha \in \mathbf{R}^N} L(w, \alpha). \tag{6.2.4}$$

Since we assume that g has a separable structure as in (6.1.2), we rewrite the saddle-point problem as

$$\min_{w \in \mathbf{R}^d} \max_{\alpha \in \mathbf{R}^N} \left\{ \frac{1}{m} \sum_{i=1}^{m} \sum_{k=1}^{n} \alpha_i^T X_{ik} w_k - \frac{1}{m} \sum_{i=1}^{m} f_i^*(\alpha_i) + \sum_{k=1}^{n} g_k(w_k) \right\}, \tag{6.2.5}$$

where $X_{ik} \in \mathbf{R}^{N_i \times d_k}$ for $k = 1, \ldots, n$ are column partitions of X_i. For convenience, we define the following notations. First, let $X = [X_1; \ldots; X_m] \in \mathbf{R}^{N \times d}$ be the overall data matrix, by stacking the X_i's vertically. Conforming to the separation of g, we also partition X into block columns $X_{:k} \in \mathbf{R}^{N \times d_k}$ for $k = 1, \ldots, n$, where each $X_{:k} = [X_{1k}; \ldots; X_{mk}]$ (stacked vertically). For consistency, we also use $X_{i:}$ to denote X_i from now on. See Figure 6.1 for an illustration.

We exploit the doubly separable structure in (6.2.5) by a doubly stochastic coordinate update algorithm outlined in Algorithm 6. Let $p = \{p_1, \ldots, p_m\}$ and $q = \{q_1, \ldots, q_n\}$ be two probability distributions. During each iteration t, we randomly pick an index $j \in \{1, \ldots, m\}$ with probability p_j, and independently pick an index $l \in \{1, \ldots, n\}$ with probability q_l. Then we compute two vectors $u_j^{(t+1)} \in \mathbf{R}^{N_j}$ and $v_l^{(t+1)} \in \mathbf{R}^{d_l}$ (details to be discussed later), and use them to update the block coordinates α_j and w_l while leaving other block coordinates unchanged. The update formulas in (6.2.6) and (6.2.7) use the proximal mappings of the (scaled) functions f_j^* and g_l respectively. We recall that the proximal mapping for any convex function $\phi : \mathbf{R}^d \to \mathbf{R} \cup \{\infty\}$ is defined as

$$\mathbf{prox}_\phi(v) \triangleq \operatorname*{argmin}_{u \in \mathbf{R}^d} \left\{ \phi(u) + \frac{1}{2} \|u - v\|^2 \right\}.$$

[1] More technically, we need to assume that each f_i is convex and lower semi-continuous so that $f_i^{**} = f_i$ [see, e.g., 155, Section 12]. It automatically holds if f_i is convex and differentiable, which we will assume later.

Algorithm 6 DSCOVR framework

Input: initial points $w^{(0)}, \alpha^{(0)}$, and step sizes σ_i for $i = 1, \ldots, m$ and τ_k for $k = 1, \ldots, n$.

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

2: pick $j \in \{1, \ldots, m\}$ and $l \in \{1, \ldots, n\}$ randomly with distributions p and q respectively.

3: compute *variance-reduced* stochastic gradients $u_j^{(t+1)}$ and $v_l^{(t+1)}$.

4: update primal and dual block coordinates:

$$\alpha_i^{(t+1)} = \begin{cases} \mathbf{prox}_{\sigma_j f_j^*}\big(\alpha_j^{(t)} + \sigma_j u_j^{(t+1)}\big) & \text{if } i = j, \\ \alpha_i^{(t)}, & \text{if } i \neq j, \end{cases} \tag{6.2.6}$$

$$w_k^{(t+1)} = \begin{cases} \mathbf{prox}_{\tau_l g_l}\big(w_l^{(t)} - \tau_l v_l^{(t+1)}\big) & \text{if } k = l, \\ w_k^{(t)}, & \text{if } k \neq l. \end{cases} \tag{6.2.7}$$

5: **end for**

There are several different ways to compute the vectors $u_j^{(t+1)}$ and $v_l^{(t+1)}$ in Step 3 of Algorithm 6. They should be the partial gradients or stochastic gradients of the bilinear coupling term in $L(w, \alpha)$ with respect to α_j and w_l respectively. Let

$$K(w, \alpha) = \alpha^T X w = \sum_{i=1}^{m} \sum_{k=1}^{n} \alpha_i^T X_{ik} w_k,$$

which is the bilinear term in $L(w, \alpha)$ without the factor $1/m$. We can use the following partial gradients in Step 3:

$$\begin{aligned} \bar{u}_j^{(t+1)} &= \frac{\partial K(w^{(t)}, \alpha^{(t)})}{\partial \alpha_j} = \sum_{k=1}^{n} X_{jk} w_k^{(t)}, \\ \bar{v}_l^{(t+1)} &= \frac{1}{m} \frac{\partial K(w^{(t)}, \alpha^{(t)})}{\partial w_l} = \frac{1}{m} \sum_{i=1}^{m} (X_{il})^T \alpha_i^{(t)}. \end{aligned} \tag{6.2.8}$$

We note that the factor $1/m$ does not appear in the first equation because it multiplies both $K(w, \alpha)$ and $f_j^*(\alpha_j)$ in (6.2.5) and hence does not appear in updating α_j. Another choice is to use

$$\begin{aligned} u_j^{(t+1)} &= \frac{1}{q_l} X_{jl} w_l^{(t)}, \\ v_l^{(t+1)} &= \frac{1}{p_j} \frac{1}{m} (X_{jl})^T \alpha_j^{(t)}, \end{aligned} \tag{6.2.9}$$

which are unbiased stochastic partial gradients, because

$$\begin{aligned} \mathbf{E}_l\big[u_j^{(t+1)}\big] &= \sum_{k=1}^{n} q_k \frac{1}{q_k} X_{jk} w_k^{(t)} = \sum_{k=1}^{n} X_{jk} w_k^{(t)} = \bar{u}_j^{(t+1)}, \\ \mathbf{E}_j\big[v_l^{(t+1)}\big] &= \sum_{i=1}^{m} p_i \frac{1}{p_i} \frac{1}{m} (X_{il})^T \alpha_i^{(t)} = \frac{1}{m} \sum_{i=1}^{m} (X_{il})^T \alpha_i^{(t)} = \bar{v}_l^{(t+1)}, \end{aligned}$$

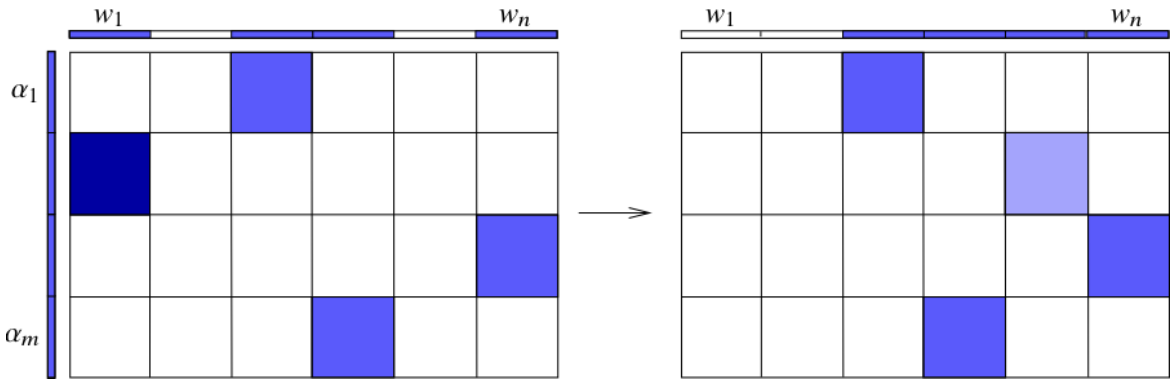

Figure 6.2: Simultaneous data and model parallelism. At any given time, each machine is busy updating one parameter block and its own dual variable. Whenever some machine is done, it is assigned to work on a random block that is not being updated.

where $\mathbf{E}_j$ and $\mathbf{E}_l$ are expectations with respect to the random indices j and l respectively.

It can be shown that, Algorithm 6 converges to a saddle point of $L(w, \alpha)$ with either choice (6.2.8) or (6.2.9) in Step 3, and with suitable *step sizes* σ_i and τ_k. It is expected that using the stochastic gradients in (6.2.9) leads to a slower convergence rate than applying (6.2.8). However, using (6.2.9) has the advantage of much less computation during each iteration. Specifically, it employs only one block matrix-vector multiplication for both updates, instead of n and m block multiplications done in (6.2.8).

More importantly, the choice in (6.2.9) is suitable for parallel and distributed computing. To see this, let $(j^{(t)}, l^{(t)})$ denote the pair of random indices drawn at iteration t (we omit the superscript (t) to simplify notation whenever there is no confusion from the context). Suppose for a sequence of consecutive iterations $t, \ldots, t+s$, there is no common index among $j^{(t)}, \ldots, j^{(t+s)}$, nor among $l^{(t)}, \ldots, l^{(t+s)}$, then these $s+1$ iterations can be done in parallel and they produce the same updates as being done sequentially. Suppose there are $s+1$ processors or machines, then each can carry out one iteration, which includes the updates in (6.2.9) as well as (6.2.6) and (6.2.7). These $s+1$ iterations are independent of each other, and in fact can be done in any order, because each only involve one primal block $w_{l^{(t)}}$ and one dual block $\alpha_{j^{(t)}}$, for both input and output (variables on the right and left sides of the assignments respectively). In contrast, the input for the updates in (6.2.8) depend on all primal and dual blocks at the previous iteration, thus cannot be done in parallel.

In practice, suppose we have m machines for solving problem (6.2.5), and each holds the data matrix $X_{i:}$ in memory and maintains the dual block α_i, for $i = 1, \ldots, m$. We assume that the number of model partitions n is larger than m, and the n model blocks $\{w_1, \ldots, w_n\}$ are stored at one or more parameter servers. In the beginning, we can randomly pick m model blocks (sampling without replacement) from $\{w_1, \ldots, w_n\}$, and assign each machine to update one of them. If machine i is assigned to update block k, then both α_i and w_k are updated, using only the matrix X_{ik}; moreover, it needs to communicate only the block w_k with the parameter server that are responsible to maintain it. Whenever one machine finishes its update, a scheduler can randomly pick another parameter block that is not currently updated by other machines, and assign it to the free machine. Therefore all machines can work in parallel, in an asynchronous,

event-driven manner. Here an event is the completion of a block update at any machine, as illustrated in Figure 6.2. We will discuss the implementation details in Section 6.7.

The idea of using doubly stochastic updates for distributed optimization in not new. It has been studied by Yun et al. [214] for solving the matrix completion problem, and by Matsushima et al. [116] for solving the saddle-point formulation of the ERM problem. Despite their nice features for parallelization, these algorithms inherit the $O(1/\sqrt{t})$ (or $O(1/t)$ with strong convexity) sublinear convergence rate of the classical stochastic gradient method. They translate into high communication and computation cost for distributed optimization. In this paper, we propose new variants of doubly stochastic update algorithms by using *variance-reduced* stochastic gradients (Step 3 of Algorithm 6). More specifically, we borrow the variance-reduction techniques from SVRG [78] and SAGA [39] to develop the DSCOVR algorithms, which enjoy fast linear rates of convergence. In the rest of this section, we summarize our theoretical results characterizing the three measures for DSCOVR: computation complexity, communication complexity, and synchronization cost. We compare them with distributed implementation of batch first-order algorithms.

6.2.1 Summary of Main Results

Throughout this paper, we use $\|\cdot\|$ to denote the standard Euclidean norm for vectors. For matrices, $\|\cdot\|$ denotes the operator (spectral) norm and $\|\cdot\|_F$ denotes the Frobenius norm. We make the following assumption regarding the optimization problem (6.1.1).

Assumption 2 *Each f_i is convex and differentiable, and its gradient is $(1/\gamma_i)$-Lipschitz continuous, i.e.,*

$$\|\nabla f_i(u) - \nabla f_i(v)\| \leq \frac{1}{\gamma_i}\|u - v\|, \quad \forall\, u, v \in \mathbf{R}^{N_i}, \quad i = 1, \ldots, m. \tag{6.2.10}$$

In addition, the regularization function g is λ-strongly convex, i.e.,

$$g(w') \geq g(w) + \xi^T(w' - w) + \frac{\lambda}{2}\|w' - w\|^2, \quad \forall\, \xi \in \partial g(w), \quad w', w \in \mathbf{R}^d.$$

Under Assumption 2, each f_i^* is γ_i-strongly convex [see, e.g., 68, Theorem 4.2.2], and $L(w, \alpha)$ defined in (6.2.1) has a unique saddle point $(w^\star, \alpha^\star)$.

The condition (6.2.10) is often referred to as f_i being $1/\gamma_i$-smooth. To simplify discussion, here we assume $\gamma_i = \gamma$ for $i = 1, \ldots, m$. Under these assumptions, each composite function $f_i(X_i w)$ has a smoothness parameter $\|X_i\|^2/\gamma$ (upper bound on the largest eigenvalue of its Hessian). Their average $(1/m)\sum_{i=1}^m f_i(X_i w)$ has a smooth parameter $\|X\|^2/(m\gamma)$, which no larger than the average of the individual smooth parameters $(1/m)\sum_{i=1}^m \|X_i\|^2/\gamma$. We define a condition number for problem (6.1.1) as the ratio between this smooth parameter and the convexity parameter λ of g:

$$\kappa_{\text{bat}} = \frac{\|X\|^2}{m\lambda\gamma} \leq \frac{1}{m}\sum_{i=1}^m \frac{\|X_{i:}\|^2}{\lambda\gamma} \leq \frac{\|X\|_{\max}^2}{\lambda\gamma}, \tag{6.2.11}$$

where $\|X\|_{\max} = \max_i\{\|X_{i:}\|\}$. This condition number is a key factor to characterize the iteration complexity of *batch* first-order methods for solving problem (6.1.1), i.e., minimizing $P(w)$.

Algorithms	Computation complexity (number of passes over data)	Communication complexity (number of vectors in $\mathbf{R}^d$)
batch first-order methods	$(1+\kappa_{\rm bat})\log(1/\epsilon)$	$m(1+\kappa_{\rm bat})\log(1/\epsilon)$
DSCOVR	$(1+\kappa_{\rm rand}/m)\log(1/\epsilon)$	$(m+\kappa_{\rm rand})\log(1/\epsilon)$
accelerated batch first-order methods	$\left(1+\sqrt{\kappa_{\rm bat}}\right)\log(1/\epsilon)$	$m\left(1+\sqrt{\kappa_{\rm bat}}\right)\log(1/\epsilon)$
accelerated DSCOVR	$\left(1+\sqrt{\kappa_{\rm rand}/m}\right)\log(1/\epsilon)$	$\left(m+\sqrt{m\cdot\kappa_{\rm rand}}\right)\log(1/\epsilon)$

Table 6.1: Computation and communication complexities of batch first-order methods and DSCOVR (for both SVRG and SAGA variants). We omit the $O(\cdot)$ notation in all entries and an extra $\log(1+\kappa_{\rm rand}/m)$ factor for accelerated DSCOVR algorithms.

Specifically, to find a w such that $P(w)-P(w^\star)\le\epsilon$, the proximal gradient method requires $O\left((1+\kappa_{\rm bat})\log(1/\epsilon)\right)$ iterations, and their accelerated variants require $O\left(\left(1+\sqrt{\kappa_{\rm bat}}\right)\log(1/\epsilon)\right)$ iterations [e.g., 14, 128, 131]. Primal-dual first order methods for solving the saddle-point problem (6.2.4) share the same complexity [19, 20].

A fundamental baseline for evaluating any distributed optimization algorithms is the distributed implementation of batch first-order methods. Let's consider solving problem (6.1.1) using the proximal gradient method. During every iteration t, each machine receives a copy of $w^{(t)}\in\mathbf{R}^d$ from a master machine (through Broadcast), and computes the local gradient $z_i^{(t)}=X_i^T\nabla f_i(X_iw^{(t)})\in\mathbf{R}^d$. Then a collective communication is invoked to compute the batch gradient $z^{(t)}=(1/m)\sum_{i=1}^m z_i^{(t)}$ at the master (Reduce). The master then takes a proximal gradient step, using $z^{(t)}$ and the proximal mapping of g, to compute the next iterate $w^{(t+1)}$ and broadcast it to every machine for the next iteration. We can also use the AllReduce operation in MPI to obtain $z^{(t)}$ at each machine without a master. In either case, the total number of passes over the data is twice the number of iterations (due to matrix-vector multiplications using both X_i and X_i^T), and the number of vectors in $\mathbf{R}^d$ sent/received across all machines is $2m$ times the number of iterations (see Table 6.1). Moreover, all communications are collective and synchronous.

Since DSCOVR is a family of randomized algorithms for solving the saddle-point problem (6.2.4), we would like to find (w,α) such that $\|w^{(t)}-w^\star\|^2+(1/m)\|\alpha^{(t)}-\alpha^\star\|^2\le\epsilon$ holds in expectation and with high probability. We list the communication and computation complexities of DSCOVR in Table 6.1, comparing them with batch first-order methods. Similar guarantees also hold for reducing the duality gap $P(w^{(t)})-D(\alpha^{(t)})$, where P and D are defined in (6.2.2) and (6.2.3) respectively.

The key quantity characterizing the complexities of DSCOVR is the condition number $\kappa_{\rm rand}$, which can be defined in several different ways. If we pick the data block i and model block k with uniform distribution, i.e., $p_i=1/m$ for $i=1,\ldots,m$ and $q_k=1/n$ for $k=1,\ldots,n$, then

$$\kappa_{\rm rand}=\frac{n\|X\|_{m\times n}^2}{\lambda\gamma},\qquad\text{where}\qquad\|X\|_{m\times n}=\max_{i,k}\|X_{ik}\|. \tag{6.2.12}$$

Algorithms	Synchronous Communication (number of vectors in $\mathbf{R}^d$)	Asynchronous Communication (equiv. number of vectors in $\mathbf{R}^d$)
DSCOVR-SVRG	$m\log(1/\epsilon)$	$\kappa_{\rm rand}\log(1/\epsilon)$
DSCOVR-SAGA	m	$(m+\kappa_{\rm rand})\log(1/\epsilon)$
accelerated DSCOVR-SVRG	$m\log(1/\epsilon)$	$\big(1+\sqrt{m\cdot\kappa_{\rm rand}}\big)\log(1/\epsilon)$
accelerated DSCOVR-SAGA	m	$\big(1+\sqrt{m\cdot\kappa_{\rm rand}}\big)\log(1/\epsilon)$

Table 6.2: Breakdown of communication complexities into synchronous and asynchronous communications for two different types of DSCOVR algorithms. We omit the $O(\cdot)$ notation and an extra $\log(1+\kappa_{\rm rand}/m)$ factor for accelerated DSCOVR algorithms.

Comparing the definition of $\kappa_{\rm bat}$ in (6.2.11), we have $\kappa_{\rm bat}\le\kappa_{\rm rand}$ because

$$\frac{1}{m}\|X\|^2\le\frac{1}{m}\sum_{i=1}^m\|X_i\|^2\le\frac{1}{m}\sum_{i=1}^m\sum_{k=1}^n\|X_{ik}\|^2\le n\|X\|_{m\times n}^2.$$

With $X_{i:}=[X_{i1}\cdots X_{im}]\in\mathbf{R}^{N_i\times d}$ and $X_{:k}=[X_{1k};\ldots;X_{mk}]\in\mathbf{R}^{N\times d_k}$, we can also define

$$\kappa'_{\rm rand}=\frac{\|X\|_{\max,F}^2}{\lambda\gamma},\qquad\text{where}\qquad\|X\|_{\max,F}=\max_{i,k}\big\{\|X_{i:}\|_F,\,\|X_{:k}\|_F\big\}.\tag{6.2.13}$$

In this case, we also have $\kappa_{\rm bat}\le\kappa'_{\rm rand}$ because $\|X\|_{\max}\le\|X\|_{\max,F}$. Finally, if we pick the pair (i,k) with non-uniform distribution $p_i=\|X_{i:}\|_F^2/\|X\|_F^2$ and $q_k=\|X_{:k}\|_F^2/\|X\|_F^2$, then we can define

$$\kappa''_{\rm rand}=\frac{\|X\|_F^2}{m\lambda\gamma}.\tag{6.2.14}$$

Again we have $\kappa_{\rm bat}\le\kappa''_{\rm rand}$ because $\|X\|\le\|X\|_F$. We may replace $\kappa_{\rm rand}$ in Tables 6.1 and 6.2 by either $\kappa'_{\rm rand}$ or $\kappa''_{\rm rand}$, depending on the probability distributions p and q and different proof techniques.

From Table 6.1, we observe similar type of speed-ups in computation complexity, as obtained by variance reduction techniques over the batch first-order algorithms for convex optimization [e.g., 4, 39, 78, 89, 91, 200], as well as for convex-concave saddle-point problems [12, 220]. Basically, DSCOVR algorithms have potential improvement over batch first-order methods by a factor of m (for non-accelerated algorithms) or $\sqrt{m}$ (for accelerated algorithms), but with a worse condition number. In the worst case, the ratio between $\kappa_{\rm rand}$ and $\kappa_{\rm bat}$ may be of order m or larger, thus canceling the potential improvements.

More interestingly, DSCOVR also has similar improvements in terms of communication complexity over batch first-order methods. In Table 6.2, we decompose the communication complexity of DSCOVR into synchronous and asynchronous communication. The decomposition turns out to be different depending on the variance reduction techniques employed: SVRG [78] versus SAGA [39]. We note that DSCOVR-SAGA essentially requires only asynchronous communication, because the synchronous communication of m vectors are only necessary for initialization with non-zero starting point.

The comparisons in Table 6.1 and 6.2 give us good understanding of the complexities of different algorithms. However, these complexities are not accurate measures of their performance in practice. For example, collective communication of m vectors in $\mathbf{R}^d$ can often be done in parallel over a spanning tree of the underlying communication network, thus only cost $\log(m)$ times (insted of m times) compared with sending only one vector. Also, for point-to-point communication, sending one vector in $\mathbf{R}^d$ altogether can be much faster than sending n smaller vectors of total length d separately. A fair comparison in term of wall-clock time on a real-world distributed computing system requires customized, efficient implementation of different algorithms. We will shed some light on timing comparisons with numerical experiments in Section 6.8.

6.2.2 Related Work

There is an extensive literature on distributed optimization. Many algorithms developed for machine learning adopt the centralized communication setting, due to the wide availability of supporting standards and platforms such as MPI, MapReduce and Spark (as discussed in the introduction). They include parallel implementations of the batch first-order and second-order methods [e.g., 24, 93, 100], ADMM [17], and distributed dual coordinate ascent [75, 109, 204].

For minimizing the average function $(1/m)\sum_{i=1}^m f_i(w)$, in the centralized setting and with only first-order oracles (i.e., gradients of f_i's or their conjugates), it has been shown that distributed implementation of accelerated gradient methods achieves the optimal convergence rate and communication complexity [7, 160]. The problem (6.1.1) we consider has the extra structure of composition with a linear transformation by the local data, which allows us to exploit simultaneous data and model parallelism using randomized algorithms and obtain improved communication and computation complexity.

Most work on asynchronous distributed algorithms exploit model parallelism in order to reduce the synchronization cost, especially in the setting with parameter servers [e.g., 8, 98, 202]. Besides, delay caused by the asynchrony can be incorporated to the step size to gain practical improvement on convergence [e.g., 2, 118, 179], though the theoretical sublinear rates remain. There are also many recent work on asynchronous parallel stochastic gradient and coordinate-descent algorithms for convex optimization [e.g., 104, 140, 149, 150, 153, 173]. When the workloads or computing power of different machines or processors are nonuniform, they may significantly increase *iteration efficiency* (number of iterations done in unit time), but often at the cost of requiring more iterations than their synchronous counterparts (due to delays and stale updates). So there is a subtle balance between iteration efficiency and iteration complexity [e.g., 58]. Our discussions in Section 6.2.1 show that DSCOVR is capable of improving both aspects.

For solving bilinear saddle-point problems with a finite-sum structure, [220] proposed a randomized algorithm that works with dual coordinate update but full primal update. Yu et al. [209] proposed a doubly stochastic algorithm that works with both primal and dual coordinate updates based on equation (6.2.8). Both of them achieved accelerated linear convergence rates, but neither can be readily applied to distributed computing. In addition, Balamurugan and Bach [12] proposed stochastic variance-reduction methods (also based on SVRG and SAGA) for solving more general convex-concave saddle point problems. For the special case with bilinear coupling, they obtained similar computation complexity as DSCOVR. However, their methods require full model updates at each iteration (even though working with only one sub-block of data), thus are

not suitable for distributed computing.

With additional assumptions and structure, such as similarity between the local cost functions at different machines or using second-order information, it is possible to obtain better communication complexity for distributed optimization; see, e.g., Reddi et al. [151], Shamir et al. [169], Zhang and Xiao [219]. However, these algorithms rely on much more computation at each machine for solving a local sub-problem at each iteration. With additional memory and preprocessing at each machine, Lee et al. [94] showed that SVRG can be adapted for distributed optimization to obtain low communication complexity.

6.3 The DSCOVR-SVRG Algorithm

From this section to Section 6.6, we present several realizations of DSCOVR using different variance reduction techniques and acceleration schemes, and analyze their convergence properties. These algorithms are presented and analyzed as sequential randomized algorithms. We will discuss how to implement them for asynchronous distributed computing in Section 6.7.

Algorithm 7 is a DSCOVR algorithm that uses the technique of SVRG [78] for variance reduction. The iterations are divided into stages and each stage has a inner loop. Each stage is initialized by a pair of vectors $\bar{w}^{(s)} \in \mathbf{R}^d$ and $\bar{\alpha}^{(s)} \in \mathbf{R}^N$, which come from either initialization (if $s = 0$) or the last iterate of the previous stage (if $s > 0$). At the beginning of each stage, we compute the batch gradients

$$\bar{u}^{(s)} = \frac{\partial}{\partial \bar{\alpha}^{(s)}} \left((\bar{\alpha}^{(s)})^T X \bar{w}^{(s)} \right) = X \bar{w}^{(s)}, \qquad \bar{v}^{(s)} = \frac{\partial}{\partial \bar{w}^{(s)}} \left(\frac{1}{m} (\bar{\alpha}^{(s)})^T X \bar{w}^{(s)} \right) = \frac{1}{m} X^T \bar{\alpha}^{(s)}.$$

The vectors $\bar{u}^{(s)}$ and $\bar{v}^{(s)}$ share the same partitions as $\alpha^{(t)}$ and $w^{(t)}$, respectively. Inside each stage s, the variance-reduced stochastic gradients are computed in (6.3.1) and (6.3.2). It is easy to check that they are unbiased. More specifically, taking expectation of $u_j^{(t+1)}$ with respect to the random index l gives

$$\mathbf{E}_l \left[u_j^{(t+1)} \right] = \bar{u}_j^{(s)} + \sum_{k=1}^{n} q_k \frac{1}{q_k} X_{jk} \left(w_k^{(t)} - \bar{w}_k^{(s)} \right) = \bar{u}_j^{(s)} + X_{j:} w^{(t)} - X_{j:} \bar{w}^{(s)} = X_{j:} w^{(t)},$$

and taking expectation of $v_l^{(t+1)}$ with respect to the random index j gives

$$\mathbf{E}_j \left[v_l^{(t+1)} \right] = \bar{v}_l^{(s)} + \sum_{i=1}^{m} p_i \frac{1}{p_i} \frac{1}{m} (X_{il})^T \left(\alpha_i^{(t)} - \bar{\alpha}_i^{(s)} \right) = \bar{v}_l^{(s)} + \frac{1}{m} (X_{:l})^T \left(\alpha^{(t)} - \bar{\alpha}^{(s)} \right) = \frac{1}{m} (X_{:l})^T \alpha^{(t)}.$$

In order to measure the distance of any pair of primal and dual variables to the saddle point, we define a weighted squared Euclidean norm on $\mathbf{R}^{d+N}$. Specifically, for any pair (w, α) where $w \in \mathbf{R}^d$ and $\alpha = [\alpha_1, \ldots, \alpha_m] \in \mathbf{R}^N$ with $\alpha_i \in \mathbf{R}^{N_i}$, we define

$$\Omega(w, \alpha) = \lambda \|w\|^2 + \frac{1}{m} \sum_{i=1}^{m} \gamma_i \|\alpha_i\|^2. \tag{6.3.3}$$

If $\gamma_i = \gamma$ for all $i = 1, \ldots, m$, then $\Omega(w, \alpha) = \lambda \|w\|^2 + \frac{\gamma}{m} \|\alpha\|^2$. We have the following theorem concerning the convergence rate of Algorithm 7.

Algorithm 7 DSCOVR-SVRG

Input: initial points $\bar{w}^{(0)}$, $\bar{\alpha}^{(0)}$, number of stages S and number of iterations per stage M.

1: **for** $s = 0, 1, 2, \ldots, S-1$ **do**

2: $\bar{u}^{(s)} = X\bar{w}^{(s)}$ and $\bar{v}^{(s)} = \frac{1}{m}X^T\bar{\alpha}^{(s)}$

3: $w^{(0)} = \bar{w}^{(s)}$ and $\alpha^{(0)} = \bar{\alpha}^{(s)}$

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

5: pick $j \in \{1, \ldots, m\}$ and $l \in \{1, \ldots, n\}$ randomly with distributions p and q respectively.

6: compute variance-reduced stochastic gradients:

$$u_j^{(t+1)} = \bar{u}_j^{(s)} + \frac{1}{q_l}X_{jl}\big(w_l^{(t)} - \bar{w}_l^{(s)}\big), \tag{6.3.1}$$

$$v_l^{(t+1)} = \bar{v}_l^{(s)} + \frac{1}{p_j}\frac{1}{m}(X_{jl})^T\big(\alpha_j^{(t)} - \bar{\alpha}_j^{(s)}\big). \tag{6.3.2}$$

7: update primal and dual block coordinates:

$$\alpha_i^{(t+1)} = \begin{cases} \mathbf{prox}_{\sigma_j f_j^*}\big(\alpha_j^{(t)} + \sigma_j u_j^{(t+1)}\big) & \text{if } i = j, \\ \alpha_i^{(t)}, & \text{if } i \neq j, \end{cases}$$

$$w_k^{(t+1)} = \begin{cases} \mathbf{prox}_{\tau_l g_l}\big(w_l^{(t)} - \tau_l v_l^{(t+1)}\big) & \text{if } k = l, \\ w_k^{(t)}, & \text{if } k \neq l. \end{cases}$$

8: **end for**

9: $\bar{w}^{(s+1)} = w^{(M)}$ and $\bar{\alpha}^{(s+1)} = \alpha^{(M)}$.

10: **end for**

Output: $\bar{w}^{(S)}$ and $\bar{\alpha}^{(S)}$.

Theorem 6.3.1 *Suppose Assumption 2 holds, and let $(w^\star, \alpha^\star)$ be the unique saddle point of $L(w, \alpha)$. Let Γ be a constant that satisfies*

$$\Gamma \geq \max_{i,k}\left\{\frac{1}{p_i}\left(1 + \frac{9\|X_{ik}\|^2}{2q_k\lambda\gamma_i}\right), \frac{1}{q_k}\left(1 + \frac{9n\|X_{ik}\|^2}{2mp_i\lambda\gamma_i}\right)\right\}. \tag{6.3.4}$$

In Algorithm 7, if we choose the step sizes as

$$\sigma_i = \frac{1}{2\gamma_i(p_i\Gamma - 1)}, \qquad i = 1, \ldots, m, \tag{6.3.5}$$

$$\tau_k = \frac{1}{2\lambda(q_k\Gamma - 1)}, \qquad k = 1, \ldots, n, \tag{6.3.6}$$

and the number of iterations during each stage satisfies $M \geq \log(3)\Gamma$, then for any $s > 0$,

$$\mathbf{E}\left[\Omega\big(\bar{w}^{(s)} - w^\star, \bar{\alpha}^{(s)} - \alpha^\star\big)\right] \leq \left(\frac{2}{3}\right)^s \Omega\big(\bar{w}^{(0)} - w^\star, \bar{\alpha}^{(0)} - \alpha^\star\big). \tag{6.3.7}$$

The proof of Theorem 6.3.1 is given in Appendix B.1. Here we discuss how to choose the parameter Γ to satisfy (6.3.4). For simplicity, we assume $\gamma_i = \gamma$ for all $i = 1, \ldots, m$.

- If we let $\|X\|_{m\times n} = \max_{i,k}\{\|X_{ik}\|\}$ and sample with the uniform distribution across both rows and columns, i.e., $p_i = 1/m$ for $i = 1, \ldots, m$ and $q_k = 1/n$ for $k = 1, \ldots, n$, then we can set

$$\Gamma = \max\{m, n\}\left(1 + \frac{9n\|X\|^2_{m\times n}}{2\lambda\gamma}\right) = \max\{m, n\}\left(1 + \frac{9}{2}\kappa_{\text{rand}}\right),$$

  where $\kappa_{\text{rand}} = n\|X\|^2_{m\times n}/(\lambda\gamma)$ as defined in (6.2.12).

- An alternative condition for Γ to satisfy is (shown in Section B.1.1 in the Appendix)

$$\Gamma \geq \max_{i,k}\left\{\frac{1}{p_i}\left(1 + \frac{9\|X_{:k}\|_F^2}{2q_k m\lambda\gamma_i}\right),\ \frac{1}{q_k}\left(1 + \frac{9\|X_{i:}\|_F^2}{2p_i m\lambda\gamma_i}\right)\right\}. \tag{6.3.8}$$

  Again using uniform sampling, we can set

$$\Gamma = \max\{m, n\}\left(1 + \frac{9\|X\|^2_{\max,F}}{2\lambda\gamma}\right) = \max\{m, n\}\left(1 + \frac{9}{2}\kappa'_{\text{rand}}\right),$$

  where $\|X\|_{\max,F} = \max_{i,k}\{\|X_{i:}\|_F, \|X_{:k}\|_F\}$ and $\kappa'_{\text{rand}} = \|X\|^2_{\max,F}/(\lambda\gamma)$ as defined in (6.2.13).

- Using the condition (6.3.8), if we choose the probabilities to be proportional to the squared Frobenius norms of the data partitions, i.e.,

$$p_i = \frac{\|X_{i:}\|_F^2}{\|X\|_F^2}, \qquad q_k = \frac{\|X_{:k}\|_F^2}{\|X\|_F^2}, \tag{6.3.9}$$

  then we can choose

$$\Gamma = \frac{1}{\min_{i,k}\{p_i, q_k\}}\left(1 + \frac{9\|X\|_F^2}{2m\lambda\gamma}\right) = \frac{1}{\min_{i,k}\{p_i, q_k\}}\left(1 + \frac{9}{2}\kappa''_{\text{rand}}\right),$$

  where $\kappa''_{\text{rand}} = \|X\|_F^2/(m\lambda\gamma)$. Moreover, we can set the step sizes as (see Appendix B.1.1)

$$\sigma_i = \frac{m\lambda}{9\|X\|_F^2}, \qquad \tau_k = \frac{m\gamma_i}{9\|X\|_F^2}.$$

- For the ERM problem (6.1.3), we assume that each loss function ϕ_j, for $j = 1, \ldots, N$, is $1/\nu$-smooth. According to (6.1.4), the smooth parameter for each f_i is $\gamma_i = \gamma = (N/m)\nu$. Let R be the largest Euclidean norm among all rows of X (or we can normalize each row to have the same norm R), then we have $\|X\|_F^2 \leq NR^2$ and

$$\kappa''_{\text{rand}} = \frac{\|X\|_F^2}{m\lambda\gamma} \leq \frac{NR^2}{m\lambda\gamma} = \frac{R^2}{\lambda\nu}. \tag{6.3.10}$$

  The upper bound $R^2/(\lambda\nu)$ is a condition number used for characterizing the iteration complexity of many randomized algorithms for ERM [e.g., 39, 78, 91, 166, 220]. In this case, using the non-uniform sampling in (6.3.9), we can set the step sizes to be

$$\sigma_i = \frac{\lambda}{9R^2}\frac{m}{N}, \qquad \tau_k = \frac{\gamma}{9R^2}\frac{m}{N} = \frac{\nu}{9R^2}. \tag{6.3.11}$$

Next we estimate the overall computation complexity of DSCOVR-SVRG in order to achieve $\mathbf{E}\big[\Omega(\bar{w}^{(s)}-w^\star, \bar{\alpha}^{(s)}-\alpha^\star)\big] \leq \epsilon$. From (6.3.7), the number of stages required is $\log\big(\Omega^{(0)}/\epsilon\big)\big/\log(3/2)$, where $\Omega^{(0)} = \Omega(\bar{w}^{(0)} - w^\star, \bar{\alpha}^{(0)} - \alpha^\star)$. The number of inner iterations within each stage is $M = \log(3)\Gamma$. At the beginning of of each stage, computing the batch gradients $\bar{u}^{(s)}$ and $\bar{v}^{(s)}$ requires going through the whole data set X, whose computational cost is equivalent to $m \times n$ inner iterations. Therefore, the overall complexity of Algorithm 7, measured by total number of inner iterations, is

$$O\left(\big(mn + \Gamma\big)\log\left(\frac{\Omega^{(0)}}{\epsilon}\right)\right).$$

To simplify discussion, we further assume $m \leq n$, which is always the case for distributed implementation (see Figure 6.2 and Section 6.7). In this case, we can let $\Gamma = n(1 + (9/2)\kappa_{\rm rand})$. Thus the above iteration complexity becomes

$$O\big(n(1 + m + \kappa_{\rm rand})\log(1/\epsilon)\big). \tag{6.3.12}$$

Since the iteration complexity in (6.3.12) counts the number of blocks X_{ik} being processed, the number of passes over the whole dataset X can be obtained by dividing it by mn, i.e.,

$$O\left(\left(1 + \frac{\kappa_{\rm rand}}{m}\right)\log(1/\epsilon)\right). \tag{6.3.13}$$

This is the computation complexity of DSCOVR listed in Table 6.1. We can replace $\kappa_{\rm rand}$ by $\kappa'_{\rm rand}$ or $\kappa''_{\rm rand}$ depending on different proof techniques and sampling probabilities as discussed above. We will address the communication complexity for DSCOVR-SVRG, including its decomposition into synchronous and asynchronous ones, after describing its implementation details in Section 6.7.

In addition to convergence to the saddle point, our next result shows that the primal-dual optimality gap also enjoys the same convergence rate, under slightly different conditions.

Theorem 6.3.2 *Suppose Assumption 2 holds, and let $P(w)$ and $D(\alpha)$ be the primal and dual functions defined in (6.2.2) and (6.2.3), respectively. Let Λ and Γ be two constants that satisfy*

$$\Lambda \;\geq\; \|X_{ik}\|_F^2\,, \qquad i = 1, \ldots, m, \quad k = 1, \ldots, n,$$

and

$$\Gamma \;\geq\; \max_{i,k}\left\{\frac{1}{p_i}\left(1 + \frac{18\Lambda}{q_k\lambda\gamma_i}\right),\; \frac{1}{q_k}\left(1 + \frac{18n\Lambda}{p_i m\lambda\gamma_i}\right)\right\}.$$

In Algorithm 7, if we choose the step sizes as

$$\sigma_i \;=\; \frac{1}{\gamma_i(p_i\Gamma - 1)}, \qquad i = 1, \ldots, m, \tag{6.3.14}$$

$$\tau_k \;=\; \frac{1}{\lambda(q_k\Gamma - 1)}, \qquad k = 1, \ldots, n, \tag{6.3.15}$$

and the number of iterations during each stage satisfies $M \geq \log(3)\Gamma$, then

$$\mathbf{E}\left[P(\bar{w}^{(s)}) - D(\bar{\alpha}^{(s)})\right] \;\leq\; \left(\frac{2}{3}\right)^s 2\Gamma\left(P(\bar{w}^{(0)}) - D(\bar{\alpha}^{(0)})\right). \tag{6.3.16}$$

The proof of Theorem 6.3.2 is given in Appendix B.2. In terms of iteration complexity or total number of passes to reach $\mathbf{E}\big[P(\bar{w}^{(s)}) - D(\bar{\alpha}^{(s)})\big] \leq \epsilon$, we need to add an extra factor of $\log(1 + \kappa_{\rm rand})$ to (6.3.12) or (6.3.13), due to the factor Γ on the right-hand side of (6.3.16).

Algorithm 8 DSCOVR-SAGA

Input: initial points $w^{(0)}, \alpha^{(0)}$, and number of iterations M.

1: $\bar{u}^{(0)} = Xw^{(0)}$ and $\bar{v}^{(0)} = \frac{1}{m} X^T \alpha^{(0)}$

2: $U_{ik}^{(0)} = X_{ik} w_k^{(0)}$, $V_{ik}^{(0)} = \frac{1}{m} (\alpha_i^{(0)})^T X_{ik}$, for all $i = 1, \ldots, m$ and $k = 1, \ldots, K$.

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

4: pick $j \in \{1, \ldots, m\}$ and $l \in \{1, \ldots, n\}$ randomly with distributions p and q respectively.

5: compute variance-reduced stochastic gradients:

$$u_j^{(t+1)} = \bar{u}_j^{(t)} - \frac{1}{q_l} U_{jl}^{(t)} + \frac{1}{q_l} X_{jl} w_l^{(t)}, \tag{6.4.1}$$

$$v_l^{(t+1)} = \bar{v}_l^{(t)} - \frac{1}{p_j} (V_{jl}^{(t)})^T + \frac{1}{p_j} \frac{1}{m} (X_{jl})^T \alpha_j^{(t)}. \tag{6.4.2}$$

6: update primal and dual block coordinates:

$$\alpha_i^{(t+1)} = \begin{cases} \mathbf{prox}_{\sigma_j \Phi_j^*}\left(\alpha_j^{(t)} + \sigma_j u_j^{(t+1)}\right) & \text{if } i = j. \\ \alpha_i^{(t)}, & \text{if } i \neq j, \end{cases}$$

$$w_k^{(t+1)} = \begin{cases} \mathbf{prox}_{\tau_l g_l}\left(w_l^{(t)} - \tau_l v_l^{(t+1)}\right) & \text{if } k = l, \\ w_k^{(t)}, & \text{if } i \neq j. \end{cases}$$

7: update averaged stochastic gradients:

$$\bar{u}_i^{(t+1)} = \begin{cases} \bar{u}_j^{(t)} - U_{jl}^{(t)} + X_{jl} w_l^{(t)} & \text{if } i = j, \\ \bar{u}_i^{(t)} & \text{if } i \neq j, \end{cases}$$

$$\bar{v}_k^{(t+1)} = \begin{cases} \bar{v}_l^{(t)} - (V_{jl}^{(t)})^T + \frac{1}{m} (X_{jl})^T \alpha_j^{(t)} & \text{if } k = l, \\ \bar{v}_k^{(t)} & \text{if } k \neq l, \end{cases}$$

8: update the table of historical stochastic gradients:

$$U_{ik}^{(t+1)} = \begin{cases} X_{jl} w_l^{(t)} & \text{if } i = j \text{ and } k = l, \\ U_{ik}^{(t)} & \text{otherwise.} \end{cases}$$

$$V_{ik}^{(t+1)} = \begin{cases} \frac{1}{m}\left((X_{jl})^T \alpha_j^{(t)}\right)^T & \text{if } i = j \text{ and } k = l, \\ V_{ik}^{(t)} & \text{otherwise.} \end{cases}$$

9: **end for**

Output: $w^{(M)}$ and $\alpha^{(M)}$.

6.4 The DSCOVR-SAGA Algorithm

Algorithm 8 is a DSCOVR algorithm that uses the techniques of SAGA [39] for variance reduction. This is a single stage algorithm with iterations indexed by t. In order to compute the

variance-reduced stochastic gradients $u_j^{(t+1)}$ and $v_l^{(t+1)}$ at each iteration, we also need to maintain and update two vectors $\bar{u}^{(t)} \in \mathbf{R}^N$ and $\bar{v}^{(t)} \in \mathbf{R}^d$, and two matrices $U^{(t)} \in \mathbf{R}^{N\times n}$ and $V^{(t)} \in \mathbf{R}^{m\times d}$. The vector $\bar{u}^{(t)}$ shares the same partition as $\alpha^{(t)}$ into m blocks, and $\bar{v}^{(t)}$ share the same partitions as $w^{(t)}$ into n blocks. The matrix $U^{(t)}$ is partitioned into $m \times n$ blocks, with each block $U_{ik}^{(t)} \in \mathbf{R}^{N_i\times 1}$. The matrix $V^{(t)}$ is also partitioned into $m \times n$ blocks, with each block $V_{ik}^{(t)} \in \mathbf{R}^{1\times d_k}$. According to the updates in Steps 7 and 8 of Algorithm 8, we have

$$\bar{u}_i^{(t)} = \sum_{k=1}^{n} U_{ik}^{(t)}, \qquad i = 1, \ldots, m, \tag{6.4.3}$$

$$\bar{v}_k^{(t)} = \sum_{i=1}^{m} \big(V_{ik}^{(t)}\big)^T, \qquad k = 1, \ldots, n. \tag{6.4.4}$$

Based on the above constructions, we can show that $u_j^{(t+1)}$ is an unbiased stochastic gradient of $(\alpha^{(t)})^T X w^{(t)}$ with respect to α_j, and $v_l^{(t+1)}$ is an unbiased stochastic gradient of $(1/m)\big((\alpha^{(t)})^T X w^{(t)}\big)$ with respect to w_l. More specifically, according to (6.4.1), we have

$$\begin{aligned}
\mathbf{E}_l\big[u_j^{(t+1)}\big] &= \bar{u}_j^{(t)} - \sum_{k=1}^{n} q_k \left(\frac{1}{q_k} U_{jk}^{(t)}\right) + \sum_{k=1}^{n} q_k \left(\frac{1}{q_k} X_{jk} w_k^{(t)}\right) \\
&= \bar{u}_j^{(t)} - \sum_{k=1}^{n} U_{jk}^{(t)} + \sum_{k=1}^{n} X_{jk} w_k^{(t)} \\
&= \bar{u}_j^{(t)} - \bar{u}_j^{(t)} + X_{j:} w^{(t)} \\
&= X_{j:} w^{(t)} = \frac{\partial}{\partial \alpha_j} \Big(\big(\alpha^{(t)}\big)^T X w^{(t)} \Big),
\end{aligned} \tag{6.4.5}$$

where the third equality is due to (6.4.3). Similarly, according to (6.4.2), we have

$$\begin{aligned}
\mathbf{E}_j\big[v_l^{(t+1)}\big] &= \bar{v}_l^{(t)} - \sum_{i=1}^{m} p_i \left(\frac{1}{p_i} (V_{il}^{(t)})^T\right) + \sum_{i=1}^{m} p_i \left(\frac{1}{p_i m} (X_{il})^T \alpha_i^{(t)}\right) \\
&= \bar{v}_l^{(t)} - \sum_{i=1}^{m} V_{il}^{(t)} + \frac{1}{m} \sum_{i=1}^{m} (X_{il})^T \alpha_i^{(t)} \\
&= \bar{v}_l^{(t)} - \bar{v}_l^{(t)} + \frac{1}{m} (X_{:l})^T \alpha^{(t)} \\
&= \frac{1}{m} (X_{:l})^T \alpha^{(t)} = \frac{\partial}{\partial w_l} \left(\frac{1}{m} \big(\alpha^{(t)}\big)^T X w^{(t)}\right),
\end{aligned} \tag{6.4.6}$$

where the third equality is due to (6.4.4).

Regarding the convergence of DSCOVR-SAGA, we have the following theorem, which is proved in Appendix B.3.

Theorem 6.4.1 *Suppose Assumption 2 holds, and let $(w^\star, \alpha^\star)$ be the unique saddle point of $L(w, \alpha)$. Let Γ be a constant that satisfies*

$$\Gamma \geq \max_{i,k} \left\{ \frac{1}{p_i} \left(1 + \frac{9\|X_{ik}\|^2}{2q_k \lambda \gamma_i}\right), \frac{1}{q_k} \left(1 + \frac{9n\|X_{ik}\|^2}{2p_i m \lambda \gamma_i}\right), \frac{1}{p_i q_k} \right\}. \tag{6.4.7}$$

If we choose the step sizes as

$$\sigma_i = \frac{1}{2\gamma_i(p_i\Gamma - 1)}, \qquad i = 1, \ldots, m, \tag{6.4.8}$$

$$\tau_k = \frac{1}{2\lambda(q_k\Gamma - 1)}, \qquad k = 1, \ldots, n, \tag{6.4.9}$$

Then the iterations of Algorithm 8 satisfy, for $t = 1, 2, \ldots$,

$$\mathbf{E}\left[\Omega\big(w^{(t)} - w^\star, \alpha^{(t)} - \alpha^\star\big)\right] \leq \left(1 - \frac{1}{3\Gamma}\right)^t \frac{4}{3}\Omega\big(w^{(0)} - w^\star, \alpha^{(0)} - \alpha^\star\big). \tag{6.4.10}$$

The condition on Γ in (6.4.7) is very similar to the one in (6.3.4), except that here we have an additional term $1/(p_iq_k)$ when taking the maximum over i and k. This results in an extra mn term in estimating Γ under uniform sampling. Assuming $m \leq n$ (true for distributed implementation), we can let

$$\Gamma = n\left(1 + \frac{9}{2}\kappa_{\text{rand}}\right) + mn.$$

According to (6.4.10), in order to achieve $\mathbf{E}\big[\Omega(w^{(t)} - w^\star, \alpha^{(t)} - \alpha^\star)\big] \leq \epsilon$, DSCOVR-SAGA needs $O\left(\Gamma \log(1/\epsilon)\right)$ iterations. Using the above expression for Γ, the iteration complexity is

$$O\big(n(1 + m + \kappa_{\text{rand}}) \log(1/\epsilon)\big), \tag{6.4.11}$$

which is the same as (6.3.12) for DSCOVR-SVRG. This also leads to the same computational complexity measured by the number of passes over the whole dataset, which is given in (6.3.13). Again we can replace κ_{rand} by κ'_{rand} or κ''_{rand} as discussed in Section 6.3. We will discuss the communication complexity of DSCOVR-SAGA in Section 6.7, after describing its implementation details.

6.5 Accelerated DSCOVR Algorithms

In this section, we develop an accelerated DSCOVR algorithm by following the "catalyst" framework [51, 101]. More specifically, we adopt the same procedure by Balamurugan and Bach [12] for solving convex-concave saddle-point problems.

Algorithm 9 proceeds in rounds indexed by $r = 0, 1, 2, \ldots$. Given the initial points $\tilde{w}^{(0)} \in \mathbf{R}^d$ and $\tilde{\alpha}^{(0)} \in \mathbf{R}^N$, each round r computes two new vectors $\tilde{w}^{(r+1)}$ and $\tilde{\alpha}^{(r+1)}$ using either the DSCOVR-SVRG or DSCOVR-SAGA algorithm for solving a regulated saddle-point problem, similar to the classical proximal point algorithm [156].

Let $\delta > 0$ be a parameter which we will determine later. Consider the following perturbed saddle-point function for round r:

$$L_\delta^{(r)}(w, a) = L(w, \alpha) + \frac{\delta\lambda}{2}\|w - \tilde{w}^{(r)}\|^2 - \frac{\delta}{2m}\sum_{i=1}^m \gamma_i\|\alpha_i - \tilde{\alpha}_i^{(r)}\|^2. \tag{6.5.1}$$

Algorithm 9 Accelerated DSCOVR

Input: initial points $\tilde{w}^{(0)}, \tilde{\alpha}^{(0)}$, and parameter $\delta > 0$.

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

2: find an approximate saddle point of (6.5.1) using one of the following two options:

• *option 1:* run Algorithm 7 with $S = \frac{2\log(2(1+\delta))}{\log(3/2)}$ and $M = \log(3)\Gamma_\delta$ to obtain

$$(\tilde{w}^{(r+1)}, \tilde{\alpha}^{(r+1)}) = \text{DSCOVR-SVRG}(\tilde{w}^{(r)}, \tilde{\alpha}^{(r)}, S, M).$$

• *option 2:* run Algorithm 8 with $M = 6\log\left(\frac{8(1+\delta)}{3}\right)\Gamma_\delta$ to obtain

$$(\tilde{w}^{(r+1)}, \tilde{\alpha}^{(r+1)}) = \text{DSCOVR-SAGA}(\tilde{w}^{(r)}, \tilde{\alpha}^{(r)}, M).$$

3: **end for**

Under Assumption 2, the function $L_\delta^{(r)}(w, a)$ is $(1+\delta)\lambda$-strongly convex in w and $(1+\delta)\gamma_i/m$-strongly concave in α_i. Let Γ_δ be a constant that satisfies

$$\Gamma_\delta \;\geq\; \max_{i,k}\left\{\frac{1}{p_i}\left(1+\frac{9\|X_{ik}\|^2}{2q_k\lambda\gamma_i(1+\delta)^2}\right),\; \frac{1}{q_k}\left(1+\frac{9n\|X_{ik}\|^2}{2p_i m\lambda\gamma_i(1+\delta)^2}\right),\; \frac{1}{p_i q_k}\right\},$$

where the right-hand side is obtained from (6.4.7) by replacing λ and γ_i with $(1+\delta)\lambda$ and $(1+\delta)\gamma_i$ respectively. The constant Γ_δ is used in Algorithm 9 to determine the number of inner iterations to run with each round, as well as for setting the step sizes. The following theorem is proved in Appendix B.4.

Theorem 6.5.1 *Suppose Assumption 2 holds, and let $(w^\star, \alpha^\star)$ be the saddle-point of $L(w, \alpha)$. With either options in Algorithm 9, if we choose the step sizes (inside Algorithm 7 or Algorithm 8) as*

$$\sigma_i \;=\; \frac{1}{2(1+\delta)\gamma_i(p_i\Gamma_\delta - 1)}, \qquad i = 1, \ldots, m, \tag{6.5.2}$$

$$\tau_k \;=\; \frac{1}{2(1+\delta)\lambda(q_k\Gamma_\delta - 1)}, \qquad k = 1, \ldots, n. \tag{6.5.3}$$

Then for all $r \geq 1$,

$$\mathbf{E}\left[\Omega\big(\tilde{w}^{(r)} - w^\star, \tilde{\alpha}^{(r)} - \alpha^\star\big)\right] \;\leq\; \left(1 - \frac{1}{2(1+\delta)}\right)^{2r}\Omega\big(\tilde{w}^{(0)} - w^\star, \tilde{\alpha}^{(0)} - \alpha^\star\big).$$

According to Theorem 6.5.1, in order to have $\mathbf{E}\big[\Omega\big(\tilde{w}^{(r)} - w^\star, \tilde{\alpha}^{(r)} - \alpha^\star\big)\big] \leq \epsilon$, we need the number of rounds r to satisfy

$$r \geq (1+\delta)\log\left(\frac{\Omega\big(\tilde{w}^{(0)} - w^\star, \tilde{\alpha}^{(0)} - \alpha^\star\big)}{\epsilon}\right).$$

Following the discussions in Sections 6.3 and 6.4, when using uniform sampling and assuming $m \leq n$, we can have

$$\Gamma_\delta = n\left(1 + \frac{9\kappa_{\text{rand}}}{2(1+\delta)^2}\right) + mn. \tag{6.5.4}$$

Then the total number of block coordinate updates in Algorithm 9 is

$$O\big((1+\delta)\Gamma_\delta \log(1+\delta)\log(1/\epsilon)\big),$$

where the $\log(1+\delta)$ factor comes from the number of stages S in option 1 and number of steps M in option 2. We hide the $\log(1+\delta)$ factor with the $\widetilde{O}$ notation and plug (6.5.4) into the expression above to obtain

$$\widetilde{O}\left(n\left((1+\delta)(1+m) + \frac{\kappa_{\text{rand}}}{(1+\delta)}\right)\log\left(\frac{1}{\epsilon}\right)\right).$$

Now we can choose δ depending on the relative size of κ_{rand} and m:

- If $\kappa_{\text{rand}} > 1 + m$, we can minimizing the above expression by choosing $\delta = \sqrt{\frac{\kappa_{\text{rand}}}{1+m}} - 1$, so that the overall iteration complexity becomes $\widetilde{O}\left(n\sqrt{m\kappa_{\text{rand}}}\log(1/\epsilon)\right)$.
- If $\kappa_{\text{rand}} \leq m + 1$, then no acceleration is necessary and we can choose $\delta = 0$ to proceed with a single round. In this case, the iteration complexity is $O(mn)$ as seen from (6.5.4).

Therefore, in either case, the total number of block iterations by Algorithm 9 can be written as

$$\widetilde{O}\left(mn + n\sqrt{m\kappa_{\text{rand}}}\log(1/\epsilon)\right). \tag{6.5.5}$$

As discussed before, the total number of passes over the whole dataset is obtained by dividing by mn:

$$\widetilde{O}\left(1 + \sqrt{\kappa_{\text{rand}}/m}\log(1/\epsilon)\right).$$

This is the computational complexity of accelerated DSCOVR listed in Table 6.1.

6.5.1 Proximal Mapping for Accelerated DSCOVR

When applying Algorithm 7 or 8 to approximate the saddle-point of (6.5.1), we need to replace the proximal mappings of $g_k(\cdot)$ and $f_i^*(\cdot)$ by those of $g_k(\cdot) + (\delta\lambda/2)\|\cdot - \tilde{w}_k^{(r)}\|^2$ and $f_i^*(\cdot) + (\delta\gamma_i/2)\|\cdot - \tilde{\alpha}_i^{(r)}\|^2$, respectively. More precisely, we replace $w_k^{(t+1)} = \mathbf{prox}_{\tau_k g_k}\big(w_k^{(t)} - \tau_k v_k^{(t+1)}\big)$ by

$$\begin{aligned} w_k^{(t+1)} &= \operatorname*{argmin}_{w_k \in \mathbf{R}^{d_k}} \left\{ g_k(w_k) + \frac{\delta\lambda}{2}\big\|w_k - \tilde{w}_k^{(r)}\big\|^2 + \frac{1}{2\tau_k}\left\|w_k - \left(w_k^{(t)} - \tau_k v_k^{(t+1)}\right)\right\|^2 \right\} \\ &= \mathbf{prox}_{\frac{\tau_k}{1+\tau_k\delta\lambda} g_k}\left(\frac{1}{1+\tau_k\delta\lambda}\left(w_k^{(t)} - \tau_k v_k^{(t+1)}\right) + \frac{\tau_k\delta\lambda}{1+\tau_k\delta\lambda}\tilde{w}_k^{(r)}\right), \end{aligned} \tag{6.5.6}$$

and replace $\alpha_i^{(t+1)} = \mathbf{prox}_{\sigma_i f_i^*}\big(\alpha_i^{(t)} + \sigma_i u_i^{(t+1)}\big)$ by

$$\begin{aligned} \alpha_i^{(t+1)} &= \operatorname*{argmin}_{\alpha_i \in \mathbf{R}^{N_i}} \left\{ f_i^*(\alpha_i) + \frac{\delta\gamma_i}{2}\big\|\alpha_i - \tilde{\alpha}_i^{(r)}\big\|^2 + \frac{1}{2\sigma_i}\left\|\alpha_i - \left(\alpha_i^{(t)} + \sigma_i u_i^{(t+1)}\right)\right\|^2 \right\} \\ &= \mathbf{prox}_{\frac{\sigma_i}{1+\sigma_i\delta\gamma_i} f_i^*}\left(\frac{1}{1+\sigma_i\delta\gamma_i}\left(\alpha_i^{(t)} + \sigma_i u_i^{(t+1)}\right) + \frac{\sigma_i\delta\gamma_i}{1+\sigma_i\delta\gamma_i}\tilde{\alpha}_i^{(r)}\right). \end{aligned} \tag{6.5.7}$$

We also examine the number of inner iterations determined by Γ_δ and how to set the step sizes. If we choose $\delta = \sqrt{\frac{\kappa_{\rm rand}}{1+m}} - 1$, then Γ_δ in (6.5.4) becomes

$$\Gamma_\delta = n\left(1 + \frac{9\kappa_{\rm rand}}{2(1+\delta)^2}\right) + mn = n\left(1 + \frac{9\kappa_{\rm rand}}{2\kappa_{\rm rand}/(m+1)}\right) + mn = 5.5(m+1)n.$$

Therefore a small constant number of passes is sufficient within each round. Using the uniform sampling, the step sizes can be estimated as follows:

$$\sigma_i = \frac{1}{2(1+\delta)\gamma_i(p_i\Gamma_\delta - 1)} \approx \frac{1}{2\sqrt{\kappa_{\rm rand}/m}\gamma_i(5.5n-1)} \approx \frac{1}{11\gamma_i n\sqrt{\kappa_{\rm rand}/m}}, \tag{6.5.8}$$

$$\tau_k = \frac{1}{2(1+\delta)\lambda(q_k\Gamma_\delta - 1)} \approx \frac{1}{2\sqrt{\kappa_{\rm rand}/m}\lambda(5.5m-1)} \approx \frac{1}{11\lambda\sqrt{m\cdot\kappa_{\rm rand}}}. \tag{6.5.9}$$

As shown by our numerical experiments in Section 6.8, the step sizes can be set much larger in practice.

6.6 Conjugate-Free DSCOVR Algorithms

A major disadvantage of primal-dual algorithms for solving problem (6.1.1) is the requirement of computing the proximal mapping of the conjugate function f_i^*, which may not admit closed-formed solution or efficient computation. This is especially the case for logistic regression, one of the most popular loss functions used in classification.

Lan and Zhou [89] developed "conjugate-free" variants of primal-dual algorithms that avoid computing the proximal mapping of the conjugate functions. The main idea is to replace the Euclidean distance in the dual proximal mapping with a Bregman divergence defined over the conjugate function itself. This technique has been used by Wang and Xiao [185] to solve structured ERM problems with primal-dual first order methods. Here we use this approach to derive conjugate-free DSCOVR algorithms. In particular, we replace the proximal mapping for the dual update

$$\alpha_i^{(t+1)} = \mathbf{prox}_{\sigma_i f_i^*}\big(\alpha_i^{(t)} + \sigma_i u_i^{(t+1)}\big) = \operatorname*{argmin}_{\alpha_i\in\mathbf{R}^{n_i}}\Big\{ f_i^*(\alpha_i) - \big\langle \alpha_i,\, u_i^{(t+1)}\big\rangle + \frac{1}{2\sigma_i}\big\|\alpha_i - \alpha_i^{(t)}\big\|^2\Big\},$$

by

$$\alpha_i^{(t+1)} = \operatorname*{argmin}_{\alpha_i\in\mathbf{R}^{n_i}}\left\{ f_i^*(\alpha_i) - \big\langle \alpha_i,\, u_i^{(t+1)}\big\rangle + \frac{1}{\sigma_i}\mathcal{B}_i\big(\alpha_i, \alpha_i^{(t)}\big)\right\}, \tag{6.6.1}$$

where $\mathcal{B}_i(\alpha_i, \alpha_i^{(t)}) = f_i^*(\alpha_i) - \big\langle \nabla f_i^*(\alpha_i^{(t)}),\, \alpha_i - \alpha_i^{(t)}\big\rangle$. The solution to (6.6.1) is given by

$$\alpha_i^{(t+1)} = \nabla f_i\big(\beta_i^{(t+1)}\big),$$

where $\beta_i^{(t+1)}$ can be computed recursively by

$$\beta_i^{(t+1)} = \frac{\beta_i^{(t)} + \sigma_i u_i^{(t+1)}}{1+\sigma_i}, \qquad t \geq 0,$$

with initial condition $\beta_i^{(0)} = \nabla f_i^*(\alpha_i^{(0)})$ [see 89, Lemma 1]. Therefore, in order to update the dual variables α_i, we do not need to compute the proximal mapping for the conjugate function f_i^*; instead, taking the gradient of f_i at some easy-to-compute points is sufficient. This conjugate-free update can be applied in Algorithms 6, 7 and 8.

For the accelerated DSCOVR algorithms, we repalce (6.5.7) by

$$\alpha_i^{(t+1)} = \operatorname*{argmin}_{\alpha_i \in \mathbf{R}^{n_i}} \left\{ f_i^*(\alpha_i) - \left\langle \alpha_i,\, u_i^{(t+1)} \right\rangle + \frac{1}{\sigma_i} \mathcal{B}_i\big(\alpha_i, \alpha_i^{(t)}\big) + \delta\gamma \mathcal{B}_i\big(\alpha_i, \tilde{\alpha}_i^{(t+1)}\big) \right\}.$$

The solution to the above minimization problem can also be written as

$$\alpha_i^{(t+1)} = \nabla f_i\big(\beta_i^{(t+1)}\big),$$

where $\beta_i^{(t+1)}$ can be computed recursively as

$$\beta_i^{(t+1)} = \frac{\beta_t^{(t)} + \sigma_i u_i^{(t+1)} + \sigma_i \delta\gamma \tilde{\beta}_i}{1 + \sigma_i + \sigma_i \delta\gamma}, \qquad t \geq 0,$$

with the initialization $\beta_i^{(0)} = \nabla f_i^*\big(\alpha_i^{(0)}\big)$ and $\tilde{\beta}_i = \nabla f_i^*\big(\tilde{\alpha}_i^{(r)}\big)$.

The convergence rates and computational complexities of the conjugate-free DSCOVR algorithms are very similar to the ones given in Sections 6.3–6.5. We omit details here, but refer the readers to Lan and Zhou [89] and Wang and Xiao [185] for related results.

6.7 Asynchronous Distributed Implementation

In this section, we show how to implement the DSCOVR algorithms presented in Sections 6.3–6.6 in a distributed computing system. We assume that the system provide both synchronous collective communication and asynchronous point-to-point communication, which are all supported by the MPI standard [122]. Throughout this section, we assume $m < n$ (see Figure 6.2).

6.7.1 Implementation of DSCOVR-SVRG

In order to implement Algorithm 7, the distributed system need to have the following components (see Figure 6.3):

- m *workers*. Each worker i, for $i = 1, \ldots, m$, stores the following local data and variables :
  - data matrix $X_{i:} \in \mathbf{R}^{N_i \times d}$.
  - vectors in $\mathbf{R}^{N_i}$: $\bar{u}_i^{(s)}$, $\alpha_i^{(t)}$, $\bar{\alpha}_i^{(s)}$.
  - vectors in $\mathbf{R}^d$: $\bar{w}^{(s)}$, $\bar{v}^{(s)}$.
  - extra buffers for computation and communication: $u_j^{(t+1)}$, $v_l^{(t+1)}$, $w_l^{(t)}$ and $w_l^{(t+1)}$.
- h *parameter servers*. Each server j stores a subset of the blocks $\big\{w_k^{(t)} \in \mathbf{R}^{d_k} : k \in \mathcal{S}_j\big\}$, where $\mathcal{S}_1, \ldots, \mathcal{S}_h$ form a partition of the set $\{1, \ldots, n\}$.
- *one scheduler*. It maintains a set of block indices $\mathcal{S}_{\text{free}} \subseteq \{1, \ldots, n\}$. At any given time, $\mathcal{S}_{\text{free}}$ contains indices of parameter blocks that are not currently updated by any worker.

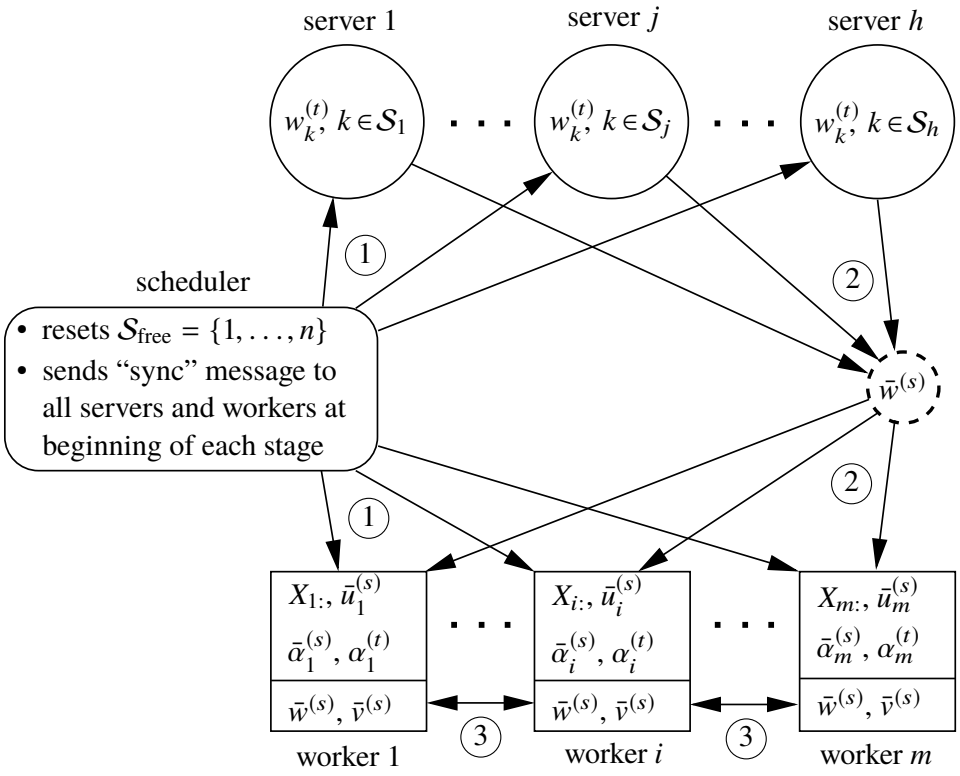

Figure 6.3: A distributed system for implementing DSCOVR consists of m workers, h parameter servers, and one scheduler. The arrows labeled with the numbers 1, 2 and 3 represent three collective communications at the beginning of each stage in DSCOVR-SVRG.

The reason for having $h > 1$ servers is not about insufficient storage for parameters, but rather to avoid the communication overload between only one server and all m workers (m can be in hundreds).

At the beginning of each stage s, the following three collective communications take place across the system (illustrated in Figure 6.3 by arrows with circled labels 1, 2 and 3):

(1) The scheduler sends a "sync" message to all servers and workers, and resets $\mathcal{S}_{\text{free}} = \{1, \ldots, n\}$.

(2) Upon receiving the "sync" message, the servers aggregate their blocks of parameters together to form $\bar{w}^{(s)}$ and send it to all workers (e.g., through the AllReduce operation in MPI).

(3) Upon receiving $\bar{w}^{(s)}$, each worker compute $\bar{u}_i^{(s)} = X_{i:}\bar{w}^{(s)}$ and $(X_{i:})^T\bar{\alpha}_i^{(s)}$, then invoke a collective communication (AllReduce) to compute $\bar{v}^{(s)} = (1/m)\sum_{i=1}^m (X_{i:})^T\bar{\alpha}_i^{(s)}$.

The number of vectors in $\mathbf{R}^d$ sent and received during the above process is $2m$, counting the communications to form $\bar{w}^{(s)}$ and $\bar{v}^{(s)}$ at m workers (ignoring the short "sync" messages).

After the collective communications at the beginning of each stage, all workers start working on the inner iterations of Algorithm 7 in parallel in an asynchronous, event-driven manner. Each worker interacts with the scheduler and the servers in a four-step loop shown in Figure 6.4. There

are always m iterations taking place concurrently (see also Figure 6.2), each may at a different phase of the four-step loop:

(1) Whenever worker i finishes updating a block k', it sends the pair (i, k') to the scheduler to request for another block to update. At the beginning of each stage, k' is not needed.

(2) When the scheduler receives the pair (i, k'), it randomly choose a block k from the list of free blocks $\mathcal{S}_{\text{free}}$ (which are not currently updated by any worker), looks up for the server j which stores the parameter block $w_k^{(t)}$ (i.e., $\mathcal{S}_j \ni k$), and then send the pair (i, k) to server j. In addition, the scheduler updates the list $\mathcal{S}_{\text{free}}$ by adding k' and deleting k.

(3) When server j receives the pair (i, k), it sends the vector $w_k^{(t)}$ to worker i, and waits for receiving the updated version $w_k^{(t+1)}$ from worker i.

(4) After worker i receives $w_k^{(t)}$, it computes the updates $\alpha_i^{(t)}$ and $w_k^{(t)}$ following steps 6-7 in Algorithm 7, and then send $w_k^{(t+1)}$ back to server j. At last, it assigns the value of k to k' and send the pair (i, k') to the scheduler, requesting the next block to work on.

The amount of point-to-point communication required during the above process is $2d_k$ float numbers, for sending and receiving $w_k^{(t)}$ and $w_k^{(t+1)}$ (we ignore the small messages for sending and receiving (i, k') and (i, k)). Since the blocks are picked randomly, the average amount of communication per iteration is $2d/n$, or equivalent to $2/n$ vectors in $\mathbf{R}^d$. According to Theorem 6.3.1, each stage of Algorithm 7 requires $\log(3)\Gamma$ inner iterations; In addition, the discussions above (6.3.12) show that we can take $\Gamma = n(1 + (9/2)\kappa_{\text{rand}})$. Therefore, the average amount of point-to-point communication within each stage is $O(\kappa_{\text{rand}})$ vectors in $\mathbf{R}^d$.

Now we are ready to quantify the communication complexity of DSCOVR-SVRG to find an ϵ-optimal solution. Our discussions above show that each stage requires collective communication of $2m$ vectors in $\mathbf{R}^d$ and asynchronous point-to-point communication of equivalently κ_{rand} such vectors. Since there are total $O(\log(1/\epsilon))$ stages, the total communication complexity is

$$O\left((m + \kappa_{\text{rand}}) \log(1/\epsilon)\right).$$

This gives the communication complexity shown in Table 6.1, as well as its decomposition in Table 6.2.

6.7.2 Implementation of DSCOVR-SAGA

We can implement Algorithm 8 using the same distributed system shown in Figure 6.3, but with some modifications described below. First, the storage at different components are different:

- m *workers.* Each worker i, for $i = 1, \ldots, m$, stores the following data and variables:
  - data matrix $X_{i:} \in \mathbf{R}^{N_i \times d}$
  - vectors in $\mathbf{R}^{N_i}$: $\alpha_i^{(t)}$, $u_i^{(t)}$, $\bar{u}_i^{(t)}$, and $U_{ik}^{(t)}$ for $k = 1, \ldots, n$.
  - vector in $\mathbf{R}^d$: $V_{i:}^{(t)} = \left[V_{i1}^{(t)} \cdots V_{in}^{(t)}\right]^T$ (which is the ith row of $V^{(t)}$, with $V_{ik}^{(t)} \in \mathbf{R}^{1 \times d_k}$).
  - buffers for communication and update of $w_k^{(t)}$ and $\bar{v}_k^{(t)}$ (both stored at some server).
- h *servers.* Each server j stores a subset of blocks $\left\{w_k^{(t)}, \bar{v}_k^{(t)} \in \mathbf{R}^{d_k} : k \in \mathcal{S}_j\right\}$, for $j = 1, \ldots, n$.

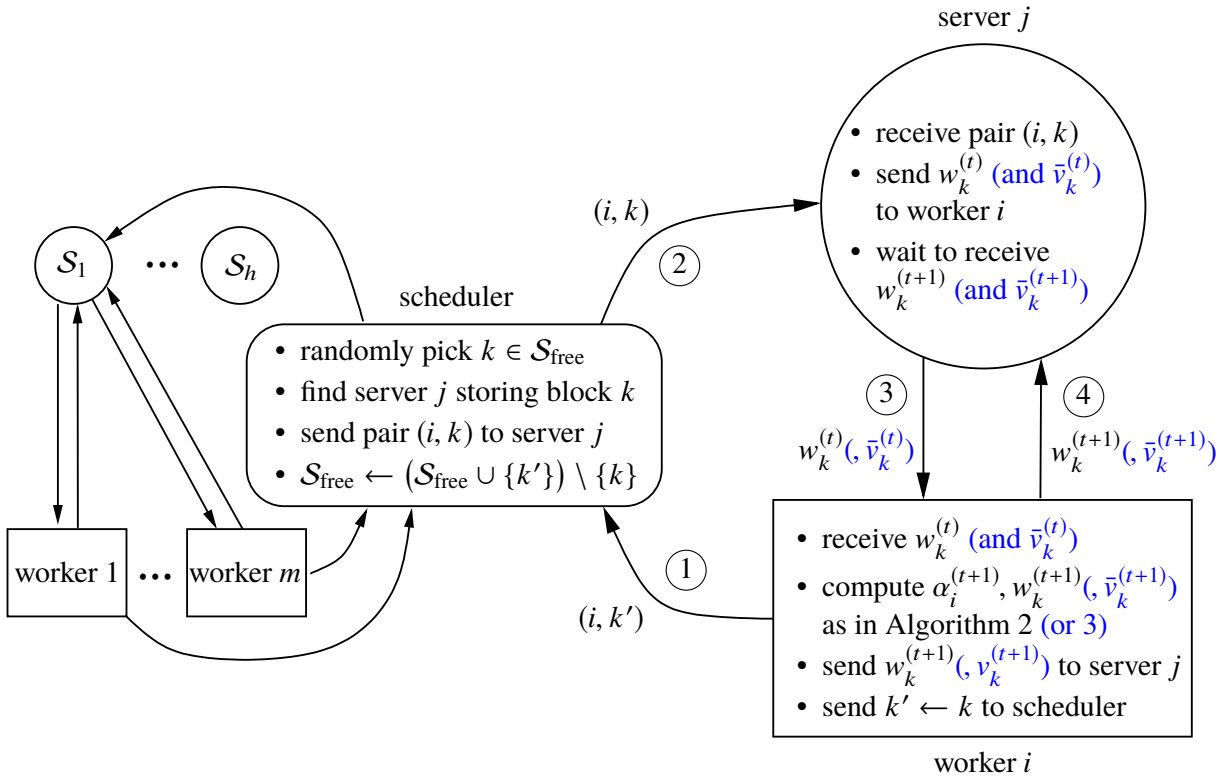

Figure 6.4: Communication and computation processes for one inner iteration of DSCOVR-SVRG (Algorithm 7). The blue texts in the parentheses are the additional vectors required by DSCOVR-SAGA (Algorithm 8). There are always m iterations taking place in parallel asynchronously, each evolving around one worker. A server may support multiple (or zero) iterations if more than one (or none) of its stored parameter blocks are being updated.

- *one scheduler*. It maintains the set of indices $\mathcal{S}_{\text{free}} \subseteq \{1, \ldots, n\}$, same as in DSCOVR-SVRG.

Unlike DSCOVR-SVRG, there is no stage-wise "sync" messages. All workers and servers work in parallel asynchronously all the time, following the four-step loops illustrated in Figure 6.4 (including blue colored texts in the parentheses). Within each iteration, the main difference from DSCOVR-SVRG is that, the server and worker need to exchange two vectors of length d_k: $w_k^{(t)}$ and $v_k^{(t)}$ and their updates. This doubles the amount of point-to-point communication, and the average amount of communication per iteration is $4/n$ vectors of length d. Using the iteration complexity in (6.4.11), the total amount of communication required (measured by number of vectors of length d) is

$$O\left((m + \kappa_{\text{rand}}) \log(1/\epsilon)\right),$$

which is the same as for DSCOVR-SVRG. However, its decomposition into synchronous and asynchronous communication is different, as shown in Table 6.2. If the initial vectors $w^{(0)} \neq 0$ or $\alpha^{(0)} \neq 0$, then one round of collective communication is required to propagate the initial conditions to all servers and workers, which reflect the $O(m)$ synchronous communication in

CPU	#cores	RAM	network	operating system
dual Intel® Xeon® processors E5-2650 (v2), 2.6 GHz	16	128 GB 1.8 GHz	10 Gbps Ethernet adapter	Windows® Server (version 2012)

Table 6.3: Configuration of each machine in the distributed computing system.

Table 6.2.

6.7.3 Implementation of Accelerated DSCOVR

Implementation of the accelerated DSCOVR algorithm is very similar to the non-accelerated ones. The main differences lie in the two proximal mappings presented in Section 6.5.1. In particular, the primal update in (6.5.6) needs the extra variable $\tilde{w}_k^{(r)}$, which should be stored at a parameter server together with $w_k^{(t)}$. We modify the four-step loops shown in Figures 6.4 as follows:

- Each parameter server j stores the extra block parameters $\{\tilde{w}_k^{(r)}, k \in \mathcal{S}_j\}$. During step (3), $\tilde{w}_k^{(r)}$ is send together with $w_k^{(t)}$ (for SVRG) or $(w_k^{(t)}, v_k^{(t)})$ (for SAGA) to a worker.
- In step (4), no update of $\tilde{w}_k^{(r)}$ is sent back to the server. Instead, whenever switching rounds, the scheduler will inform each server to update their $\tilde{w}_k^{(r)}$ to the most recent $w_k^{(t)}$.

For the dual proximal mapping in (6.5.7), each worker i needs to store an extra vector $\tilde{\alpha}_i^{(r)}$, and reset it to the most recent $\alpha_i^{(t)}$ when moving to the next round. There is no need for additional synchronization or collective communication when switching rounds in Algorithm 9. The communication complexity (measured by the number of vectors of length d sent or received) can be obtained by dividing the iteration complexity in (6.5.5) by n, i.e., $O\big((m + \sqrt{m\kappa_{\text{rand}}})\log(1/\epsilon)\big)$, as shown in Table 6.1.

Finally, in order to implement the conjugate-free DSCOVR algorithms described in Section 6.6, each worker i simply need to maintain and update an extra vector $\beta_i^{(t)}$ locally.

6.8 Experiments

In this section, we present numerical experiments on an industrial distributed computing system. This system has hundreds of computers connected by high speed Ethernet in a data center. The hardware and software configurations for each machine are listed in Table 6.3. We implemented all DSCOVR algorithms presented in this paper, including the SVRG and SAGA versions, their accelerated variants, as well as the conjugate-free algorithms. All implementations are written in C++, using MPI for both collective and point-to-point communications (see Figures 6.3 and 6.4 respectively). On each worker machine, we also use OpenMP [135] to exploit the multi-core architecture for parallel computing, including sparse matrix-vector multiplications and vectorized function evaluations.

Implementing the DSCOVR algorithms requires $m + h + 1$ machines, among them m are workers with local datasets, h are parameter servers, and one is a scheduler (see Figure 6.3). We

Dataset	#instances (N)	#features (d)	#nonzeros
`rcv1-train`	677,399	47,236	49,556,258
`webspam`	350,000	16,609,143	1,304,697,446
`splice-site`	50,000,000	11,725,480	166,167,381,622

Table 6.4: Statistics of three datasets. Each feature vector is normalized to have unit norm.

focus on solving the ERM problem (6.1.3), where the total of N training examples are evenly partitioned and stored at m workers. We partition the d-dimensional parameters into n subsets of roughly the same size (differ at most by one), where each subset consists of randomly chosen coordinates (without replacement). Then we store the n subsets of parameters on h servers, each getting either $\lfloor n/h \rfloor$ or $\lceil n/h \rceil$ subsets. As described in Section 6.7, we make the configurations to satisfy $n > m > h \geq 1$.

For DSCOVR-SVRG and DSCOVR-SAGA, the step sizes in (6.3.11) are very conservative. In the experiments, we replace the coefficient $1/9$ by two tuning parameter η_{d} and η_{p} for the dual and primal step sizes respectively, i.e.,

$$\sigma_i = \eta_{\mathrm{d}} \frac{\lambda}{R^2} \cdot \frac{m}{N}, \qquad \tau_k = \eta_{\mathrm{p}} \frac{\nu}{R^2}. \tag{6.8.1}$$

For the accelerated DSCOVR algorithms, we use $\kappa_{\mathrm{rand}} = R^2/(\lambda\nu)$ as shown in (6.3.10) for ERM. Then the step sizes in (6.5.8) and (6.5.9), with $\gamma_i = (m/N)\nu$ and a generic constant coefficient η, become

$$\sigma_i = \frac{\eta_{\mathrm{d}}}{nR} \sqrt{\frac{m\lambda}{\nu}} \cdot \frac{m}{N}, \qquad \tau_k = \frac{\eta_{\mathrm{p}}}{R} \sqrt{\frac{\nu}{m\lambda}}. \tag{6.8.2}$$

For comparison, we also implemented the following first-order methods for solving problem 6.1.1:

- PGD: parallel implementation of the Proximal Gradient Descent method (using synchronous collective communication over m machines). We use the adaptive line search procedure proposed in Nesterov [131], and the exact form used is Algorithm 2 in Lin and Xiao [102].
- APG: parallel implementation of the Accelerated Proximal Gradient method [128, 131]. We use a similar adaptive line search scheme to the one for PGD, and the exact form used (with strong convexity) is Algorithm 4 in Lin and Xiao [102].
- ADMM: the Alternating Direction Method of Multipliers. We use the regularized consensus version in Boyd et al. [17, Section 7.1.1]. For solving the local optimization problems at each node, we use the SDCA method [166].
- CoCoA+: the adding version of CoCoA in Ma et al. [109]. Following the suggestion in Ma et al. [110], we use a randomized coordinate descent algorithm [129, 152] for solving the local optimization problems.

These four algorithms all require m workers only. Specifically, we use the AllReduce call in MPI for the collective communications so that a separate master machine is not necessary.

We conducted experiments on three binary classification datasets obtained from the collection maintained by Fan and Lin [49]. Table 6.4 lists their sizes and dimensions. In our experiments, we used two configurations: one with $m = 20$ and $h = 10$ for two relatively small datasets,

`rcv1-train` and `webspam`, and the other with $m = 100$ and $h = 20$ for the large dataset `splice-site`.

For `rcv1-train`, we solve the ERM problem (6.1.3) with a smoothed hinge loss defined as

$$\phi_j(t) = \begin{cases} 0 & \text{if } y_j t \geq 1, \\ \frac{1}{2} - y_j t & \text{if } y_j t \leq 0, \\ \frac{1}{2}(1 - y_j t)^2 & \text{otherwise}, \end{cases} \qquad \text{and} \qquad \phi_j^*(\beta) = \begin{cases} y_j \beta + \frac{1}{2}\beta^2 & \text{if } -1 \leq y_j \beta \leq 0, \\ +\infty & \text{otherwise.} \end{cases}$$

for $j = 1, \ldots, N$. This loss function is 1-smooth, therefore $\nu = 1$; see discussion above (6.3.10). We use the ℓ_2 regularization $g(w) = (\lambda/2)\|w\|^2$. Figures 6.5 and 6.6 show the reduction of the primal objective gap $P(w^{(t)}) - P(w^\star)$ by different algorithms, with regularization parameter $\lambda = 10^{-4}$ and $\lambda = 10^{-6}$ respectively. All started from the zero initial point. Here the N examples are randomly shuffled and then divided into m subsets. The labels SVRG and SAGA mean DSCOVR-SVRG and DSCOVR-SAGA, respectively, and A-SVRG and A-SAGA are their accelerated versions.

Since PGD and APG both use adaptive line search, there is no parameter to tune. For ADMM, we manually tuned the penalty parameter ρ [see 17, Section 7.1.1] to obtain good performance: $\rho = 10^{-5}$ in Figure 6.5 and $\rho = 10^{-6}$ in Figure 6.6. For CoCoA+, two passes over the local datasets using a randomized coordinate descent method are sufficient for solving the local optimization problem (more passes do not give meaningful improvement). For DSCOVR-SVRG and SAGA, we used $\eta_{\mathrm{p}} = \eta_{\mathrm{d}} = 20$ to set the step sizes in (6.8.1). For DSCOVR-SVRG, each stage goes through the whole dataset 10 times, i.e., the number of inner iterations in Algorithm 7 is $M = 10mn$. For the accelerated DSCOVR algorithms, better performance are obtained with small periods to update the proximal points and we set it to be every 0.2 passes over the dataset, i.e., $0.2mn$ inner iterations. For accelerated DSCOVR-SVRG, we set the stage period (for variance reduction) to be $M = mn$, which is actually longer than the period for updating the proximal points.

From Figures 6.5 and 6.6, we observe that the two distributed algorithms based on model averaging, ADMM and CoCoA+, converges relatively fast in the beginning but becomes very slow in the later stage. Other algorithms demonstrate more consistent linear convergence rates. For $\lambda = 10^{-4}$, the DSCOVR algorithms are very competitive compared with other algorithms. For $\lambda = 10^{-6}$, the non-accelerated DSCOVR algorithms become very slow, even after tuning the step sizes. But the accelerated DSCOVR algorithms are superior in terms of both number of passes over data and wall-clock time (with adjusted step size coefficient $\eta_{\mathrm{p}} = 10$ and $\eta_{\mathrm{d}} = 40$).

For ADMM and CoCoA+, each marker represents the finishing of one iteration. It can be seen that they are mostly evenly spaced in terms of number of passes over data, but have large variations in terms of wall-clock time. The variations in time per iteration are due to resource sharing with other jobs running simultaneously on the distributed computing cluster. Even if we have exclusive use of each machine, sharing communications with other jobs over the Ethernet is unavoidable. This reflects the more realistic environment in cloud computing.

For the `webspam` dataset, we solve the ERM problem with logistic loss $\phi_j(t) = \log(1 + \exp(-y_j t))$ where $y_j \in \{\pm 1\}$. The logistic loss is $1/4$-smooth, so we have $\nu = 4$. Since the proximal mapping of its conjugate ϕ_j^* does not have a closed-form solution, we used the

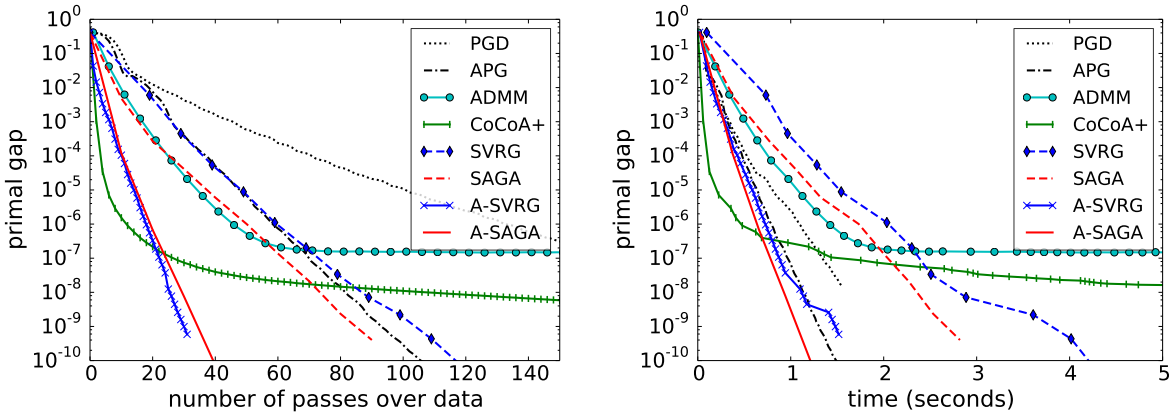

Figure 6.5: `rcv1-train`: smoothed-hinge loss, $\lambda = 10^{-4}$, randomly shuffled, $m = 20$, $n = 37$, $h = 10$.

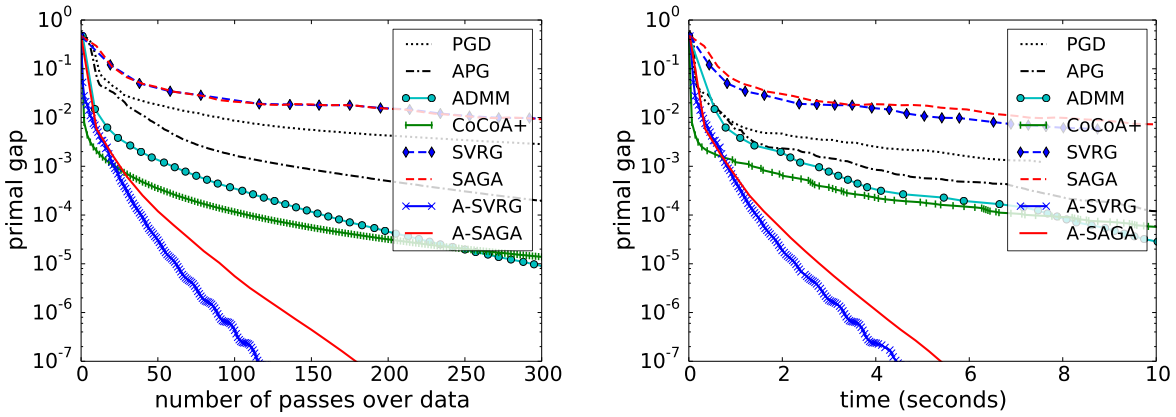

Figure 6.6: `rcv1-train`: smoothed-hinge loss, $\lambda = 10^{-6}$, randomly shuffled, $m = 20$, $n = 37$, $h = 10$.

conjugate-free DSCOVR algorithms described in Section 6.6. Figures 6.7 and 6.8 shows the reduction of primal objective gap by different algorithms, for $\lambda = 10^{-4}$ and $\lambda = 10^{-6}$ respectively. Here the starting point is no longer the all-zero vectors. Instead, each machine i first computes a local solution by minimizing $f_i(X_i w) + g(w)$, and then compute their average using an AllReduce operation. Each algorithm starts from this average point. This averaging scheme has been proven to be very effective to warm start distributed algorithms for ERM [221]. In addition, it can be shown that when starting from the zero initial point, the first step of CoCoA+ computes exactly such an averaged point.

From Figures 6.7 and 6.8, we again observe that CoCoA+ has very fast convergence in the beginning but converges very slowly towards higher precision. The DSCOVR algorithms, especially the accelerated variants, are very competitive in terms of both number of iterations and wall-clock time.

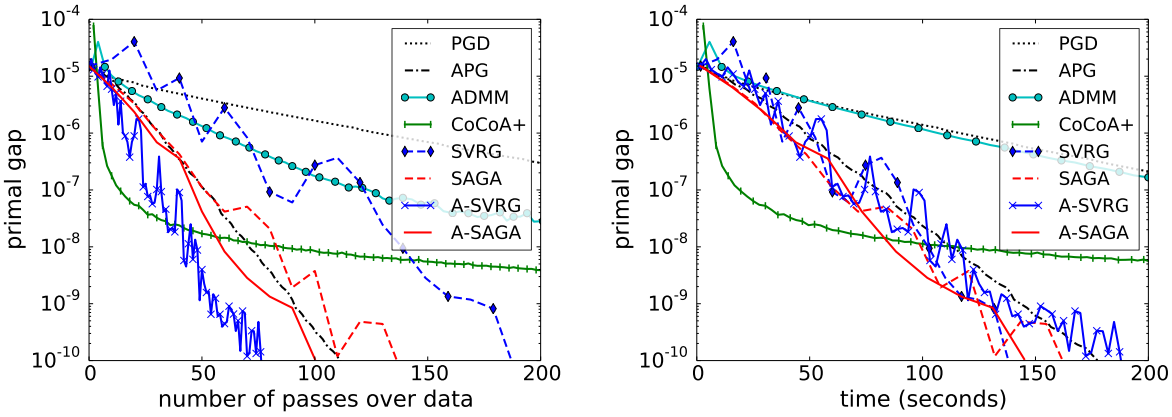

Figure 6.7: webspam: logistic regression, $\lambda = 10^{-4}$, randomly shuffled, $m = 20$, $n = 50$, $h = 10$.

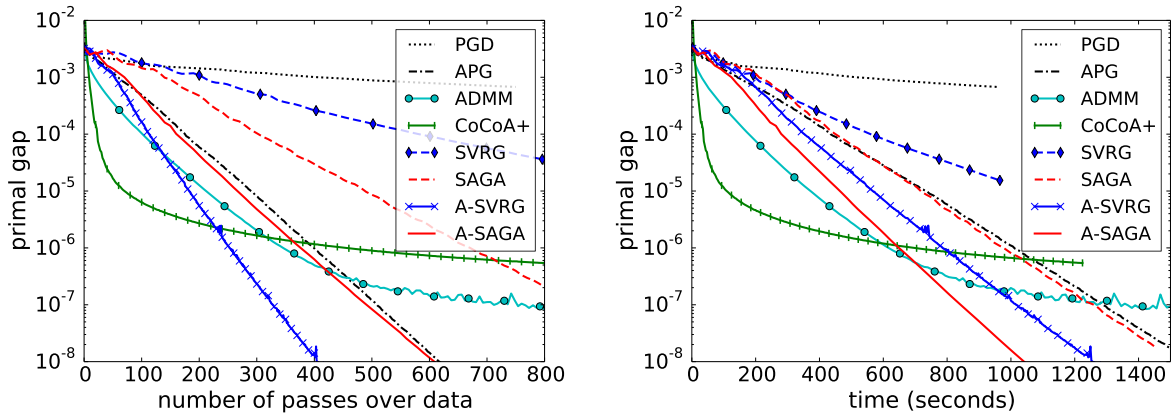

Figure 6.8: webspam: logistic regression, $\lambda = 10^{-6}$, randomly shuffled, $m = 20$, $n = 50$, $h = 10$.

In order to investigate the fast initial convergence of CoCoA+ and ADMM, we repeated the experiments on webspam without random shuffling. More specifically, we sorted the N examples by their labels, and then partitioned them into m subsets sequentially. That is, most of the machines have data with only $+1$ or -1 labels, and only one machine has mixed ± 1 examples. The results are shown in Figures 6.9 and 6.10. Now the fast initial convergence of CoCoA+ and ADMM disappeared. In particular, CoCoA+ converges with very slow linear rate. This shows that statistical properties of random shuffling of the dataset is the main reason for the fast initial convergence of model-averaging based algorithms such as CoCoA+ and ADMM [see, e.g., 221].

On the other hand, this should not have any impact on PGD and APG, because their iterations are computed over the whole dataset, which is the same regardless of random shuffling or sorting. The differences between the plots for PGD and APG in Figures 6.7 and 6.9 (also for Figures 6.8 and 6.10) are due to different initial points computed through averaging local solutions, which does depends on the distribution of data at different machines.

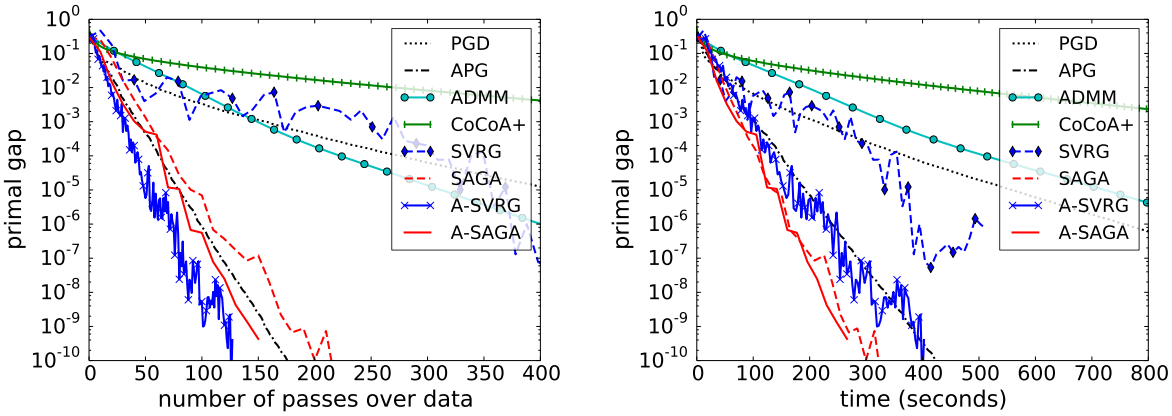

Figure 6.9: `webspam`: logistic regression, $\lambda = 10^{-4}$, sorted labels, $m = 20$, $n = 50$, $h = 10$.

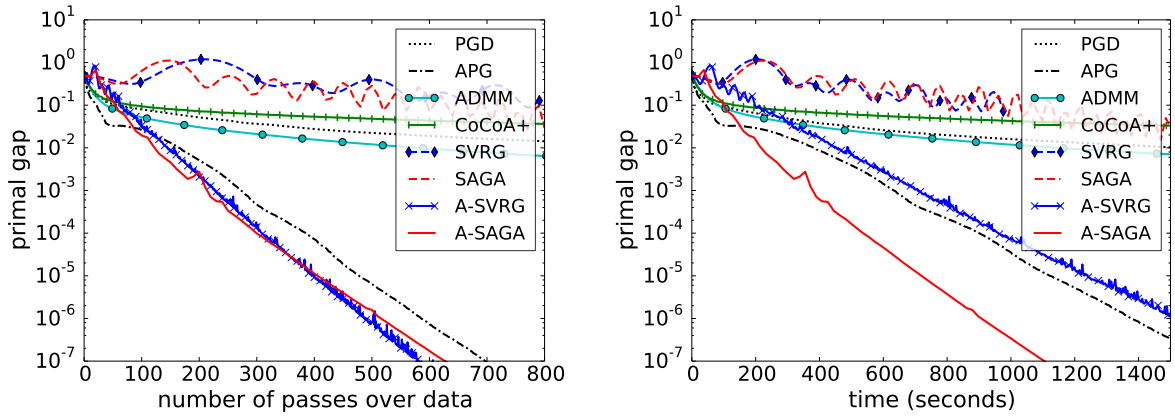

Figure 6.10: `webspam`: logistic regression, $\lambda = 10^{-6}$, sorted labels, $m = 20$, $n = 50$, $h = 10$.

Different ways for splitting the data over the m workers also affect the DSCOVR algorithms. In particular, the non-accelerated DSCOVR algorithms become very slow, as shown in Figures 6.9 and 6.10. However, the accelerated DSCOVR algorithms are still very competitive against the adaptive APG. The accelerated DSCOVR-SAGA algorithm performs best. In fact, the time spent by accelerated DSCOVR-SAGA should be even less than shown in Figures 6.9 and 6.10. Recall that other than the initialization with non-zero starting point, DSCOVR-SAGA is completely asynchronous and does not need any collective communication (see Section 6.7.2). However, in order to record the objective function for the purpose of plotting its progress, we added collective communication and computation to evaluate the objective value for every 10 passes over the data. For example, in Figure 6.10, such extra collective communications take about 160 seconds (about 15% of total time) for accelerated DSCOVR-SAGA, which can be further deducted from the horizontal time axis.

Finally, we conducted experiments on the `splice-site` dataset with 100 workers and 20 parameter servers. The results are shown in Figure 6.11. Here the dataset is again randomly shuffled and evenly distributed to the workers. The relative performance of different algorithms are similar to those for the other datasets.

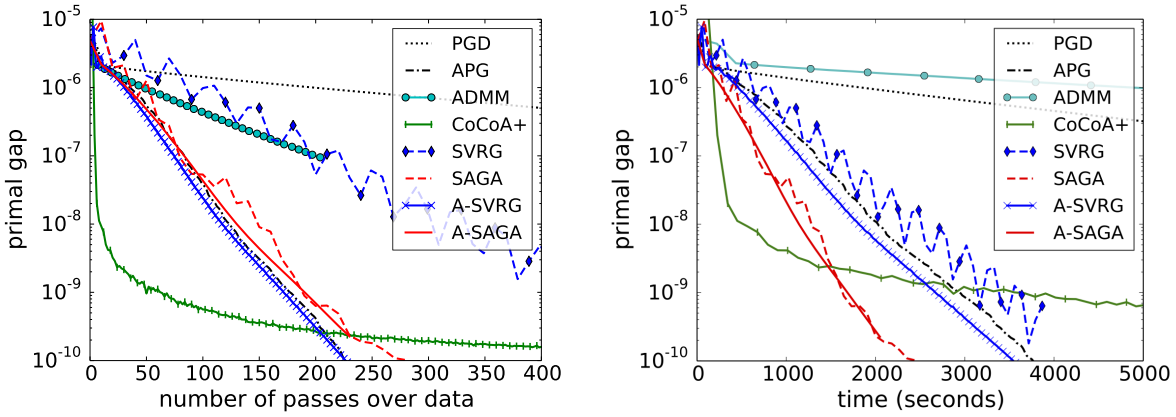

Figure 6.11: `splice-site`: logistic loss, $\lambda = 10^{-6}$. randomly shuffled, $m = 100$, $n = 150$, $h = 20$.

6.9 Discussion

We proposed a class of DSCOVR algorithms for asynchronous distributed optimization of large linear models with convex loss functions. They avoid dealing with delays and stale updates in an asynchronous, event-driven environment by exploiting simultaneous data and model parallelism. Compared with other first-order distributed algorithms, DSCOVR may require less amount of overall communication and computation, and especially much less or no synchronization. These conclusions are well supported by our computational experiments on a distributed computing cluster.

We note that there is still some gap between theory and practice. In our theoretical analysis, we assume that the primal and dual block indices in different iterations of DSCOVR are i.i.d. random variables, sampled sequentially with replacement. But the parallel implementations described in Section 6.7 impose some constraints on how they are generated. In particular, the parameter block to be updated next is randomly chosen from the set of blocks that are not being updated by any worker simultaneously, and the next worker available is event-driven, depending on the loads and processing power of different workers as well as random communication latency. These constraints violate the i.i.d. assumption, but our experiments show that they still lead to very competitive performance. Intuitively some of them can be potentially beneficial, reminiscent of the practical advantage of sampling without replacement over sampling with replacement in randomized coordinate descent methods [e.g., 166]. This is an interesting topic worth future study.

In our experiments, the parallel implementation of Nesterov's accelerated gradient method (APG) is very competitive on all the datasets we tried and for different regularization parameters used. In addition to the theoretical justifications in Arjevani and Shamir [7] and Scaman et al. [160], the adaptive line-search scheme turns out to be critical for its good performance in practice. The accelerated DSCOVR algorithms demonstrated comparable or better performance than APG in the experiments, but need careful tuning of the constants in their step size formula. On one hand, it supports our theoretical results that DSCOVR is capable of outperforming other first-

order algorithms including APG, in terms of both communication and computation complexity (see Section 6.2.1). On the other hand, there are more to be done in order to realize the full potential of DSCOVR in practice. In particular, we plan to follow the ideas in [185] to develop adaptive schemes that can automatically tune the step size parameters, as well as exploit strong convexity from data.

Chapter 7

Concluding Remarks

7.1 Conclusions

The goal of this thesis is to improve the efficiency and effectiveness of large scale machine learning. We break down the task into the following questions: 1) How to efficiently train a machine learning model? 2) How to speed up inference after the model is trained? 3) How to make the model generalize better? We approach them in a principled way from two perspectives – models and algorithms. The contributions of the thesis are restated below.

- **Proposed Models**:
  - **Recurrent model with selective reading**: Based on the assumption that texts are usually redundant and inspired by the skimming reading behavior of humans, in Chapter 2, we propose a model LSTM-Jump, which can read text while skipping irrelevant information if needed. Our model is the first of this kind to mimic the skimming, and thus can be up to 6 times faster than a sequential model when applied to real reading comprehension tasks.
  - **Recurrence-free model for parallel computation**: Noting that recurrent neural networks are often slow for both training and inference due to their sequential nature, in Chapter 3, we propose a new text encoding architecture QANet, which consists exclusively of convolution and self-attention, where convolution captures local interactions and self-attention models global interactions. On the SQuAD question answering dataset, our model is up to 13x and 9x faster in training and inference respectively, compared with recurrent models. We also propose a generic data augmentation technique called backtranslation, combined with which, QANet stayed on top of the competitive SQuAD leaderboard for half a year, and superseded the human performance in terms of exact match.
- **Proposed Algorithms**:
  - **Block-normalized gradient algorithm**: In view of the gradient vanishing problem in deep neural networks, Chapter 4 proposes a generic and simple strategy for utilizing stochastic gradient information in optimization. The technique essentially contains two consecutive steps in each iteration: 1) computing and normalizing each block

(layer) of the mini-batch stochastic gradient; 2) selecting appropriate step size to update the decision variable (parameter) towards the negative of the block-normalized gradient. Extensive empirical studies on various non-convex neural network optimization problems, including multi layer perceptron, convolution neural networks and recurrent networks indicate the block-normalized gradient can help avoid the vanishing gradient issue and accelerate the training of neural networks.

- **Randomized primal-dual coordinate methods**: In Chapter 5 and 6, we propose two primal-dual algorithms for the classical empirical risk minimization problem. While the first method, DSPDC, is designed for the single-machine environment, the second one, DSCOVR, can work well in the distributed parameter server framework. Both methods enjoy a fast linear convergence rate theoretically and show delighting empirical performance on various tasks, such as multitask large margin nearest neighbor, matrix risk minimization among many others.

7.2 Future Directions

In this thesis, we have taken several steps in models and algorithms to boost the efficiency and efficacy of large scale learning. In the future, there are several interesting directions going forward worth investigating, which we highlight below.

1. Theory: Bridging the gap between theory and practice. In this thesis, we show that the proposed algorithms have provable convergence guarantee under several assumptions, e.g., convexity of the objective functions. However, in practice, the objective might be highly non-convex, especially in the scenario of deep neural network training. While our algorithms such as normalization gradient method still work well in the non-convex setting, we still have little clue why it is the case. On the other hand, although our algorithms focuses on how to accelerate the training, it not clear to us why they also improve the generalization. How to develop new theories to bridge those gaps still pose great challenges and is well worth investigating.

2. Algorithm: Further scaling up model training. Evidences have shown that more data in training can usually lead to better generalization performance, which, for example, can be seen in accuracy gain by the back-translation technique in QANet. However, to fit larger data, we have to train bigger models, which presents a few research questions: 1) How could we increase the model size in terms of depth and width, but still ensure that its training converges? 2) How could we train a huge model that could not fit in the modern hardware?

3. Application: Human level language understanding. We have proposed several methods in this thesis to efficiently process natural language, one of which, i.e., QANet, has even achieved the human performance on a specific dataset. However, the error analysis of QANet show that we are still far away from having a model that can achieve human-level language understanding. There are at least three remaining problems we have to address to enable the genuine human intelligence. 1) How to digest extremely long texts, e.g., a book? 2) How to incorporate human

knowledge, e.g., commonsense, to the model? 3) How to equip the model with logical reasoning power? A perfect touchstone to benchmark the language understanding capability of the model might be the United States Medical Licensing Examination (USMLE), which is notoriously difficult, due to its extensivity on various medical subjects and the requirement on rigorous logical reasoning based on massive medical knowledge and the complicated patient situation.

Part III

Appendices

Appendix A

Convergence Analysis for DSPDC

In this section, we provide the detailed proof of the main theoretical results in Section 5.4.

A.1 Some technical lemmas

In order to prove Theorem 5.4.1, we first present the following two technical lemmas which are extracted but extended from [220]. In particular, the second inequality in both lemmas are given in [220] while the first inequality is new and is the key to prove the convergence in objective gap. These lemmas establish the relationship between two consecutive iterates, $(x^{(t)}, y^{(t)})$ and $(x^{(t+1)}, y^{(t+1)})$.

Lemma 1 *Given any $\bar{x} \in \mathbb{R}^p$ and $v \in \mathbb{R}^p$, if we uniformly and randomly choose a set of indices $J \subset \{1, 2, \ldots, p\}$ with $|J| = q$ and solve an $\hat{x} \in \mathbb{R}^p$ with*

$$\hat{x}_j = \begin{cases} \operatorname{argmin}_{\alpha \in \mathbb{R}} \left\{ \frac{1}{n} v_j \alpha + g_j(\alpha) + \frac{1}{2\tau}(\alpha - \bar{x}_j)^2 \right\} & \text{if } j \in J \\ \bar{x}_j & \text{if } j \notin J, \end{cases} \tag{A.1.1}$$

then, any $x \in \mathbb{R}^p$, we have

$$\begin{aligned} & \left(\frac{p}{2q\tau} + \frac{(p-q)\lambda}{2q} \right) \|x - \bar{x}\|^2 + \frac{p-q}{q} \left(g(\bar{x}) - g(x) \right) \\ \geq & \left(\frac{p}{2q\tau} + \frac{p\lambda}{2q} \right) \mathbb{E}\|\hat{x} - x\|^2 + \frac{p}{2q\tau} \mathbb{E}\|\hat{x} - \bar{x}\|^2 + \frac{p}{q} \mathbb{E}\left(g(\hat{x}) - g(x) \right) + \frac{1}{n} \mathbb{E} \left\langle v, \bar{x} + \frac{p}{q}(\hat{x} - \bar{x}) - x \right\rangle \end{aligned}$$

and

$$\begin{aligned} & \left(\frac{p}{2q\tau} + \frac{(p-q)\lambda}{q} \right) \|x^\star - \bar{x}\|^2 \\ \geq & \left(\frac{p}{2q\tau} + \frac{p\lambda}{q} \right) \mathbb{E}\|\hat{x} - x^\star\|^2 + \frac{p}{2q\tau} \mathbb{E}\|\hat{x} - \bar{x}\|^2 + \frac{1}{n} \mathbb{E} \left\langle v - A^T y^\star, \bar{x} + \frac{p}{q}(\hat{x} - \bar{x}) - x^\star \right\rangle, \end{aligned}$$

where the expectation $\mathbb{E}$ is taken over J.

Proof: We prove the first conclusion first. Let $\tilde{x}$ defined as

$$\tilde{x} = \operatorname*{argmin}_{x\in\mathbb{R}^n} \left\{ \frac{1}{n} v^T x + g(x) + \frac{1}{2\tau}\|x - \bar{x}\|^2 \right\}.$$

Therefore, according to (A.1.1), $\hat{x}_j = \tilde{x}_j$ if $j \in J$ and $\hat{x}_j = \bar{x}_j$ if $j \notin J$. Due to the decomposable structure (5.1.2) of $g(x)$, each coordinate $\tilde{x}_j$ of $\tilde{x}$ can be solved independently. Since g_j is λ-strongly convex, the optimality of $\tilde{x}_j$ implies that, for any $x_j \in \mathbb{R}$,

$$\frac{v_j x_j}{n} + g_j(x_j) + \frac{(x_j - \bar{x}_j)^2}{2\tau} \geq \frac{v_j \tilde{x}_j}{n} + g_j(\tilde{x}_j) + \frac{(\tilde{x}_j - \bar{x}_j)^2}{2\tau} + \left(\frac{1}{2\tau} + \frac{\lambda}{2}\right)(\tilde{x}_j - x_j)^2. \quad \text{(A.1.2)}$$

Since each index j is contained in J with a probability of $\frac{q}{p}$, we have the following equalities

$$\begin{aligned}
\mathbb{E}(\hat{x}_j - x_j)^2 &= \frac{q}{p}(\tilde{x}_j - x_j)^2 + \frac{p-q}{p}(\bar{x}_j - x_j)^2, && \text{(A.1.3)}\\
\mathbb{E}(\hat{x}_j - \bar{x}_j)^2 &= \frac{q}{p}(\tilde{x}_j - \bar{x}_j)^2, && \text{(A.1.4)}\\
\mathbb{E}\hat{x}_j &= \frac{q}{p}\tilde{x}_j + \frac{p-q}{p}\bar{x}_j, && \text{(A.1.5)}\\
\mathbb{E}g_j(\hat{x}_j) &= \frac{q}{p}g_j(\tilde{x}_j) + \frac{p-q}{p}g_j(\bar{x}_j). && \text{(A.1.6)}
\end{aligned}$$

Using these equalities, we can represent all the terms in (A.1.2) involving $\tilde{x}_j$ by the terms that only contains $\hat{x}_j$, $\bar{x}_j$ and x_j. By doing so and organizing terms, we obtain

$$\begin{aligned}
&\left(\frac{p}{2q\tau} + \frac{(p-q)\lambda}{2q}\right)(x_j - \bar{x}_j)^2 + \frac{p-q}{q}\left(g_j(\bar{x}_j) - g_j(x_j)\right) - \frac{1}{n}v_j\left(\bar{x}_j + \frac{p}{q}(\mathbb{E}\hat{x}_j - \bar{x}_j) - x_j\right)\\
\geq\ &\left(\frac{p}{2q\tau} + \frac{p\lambda}{2q}\right)\mathbb{E}(\hat{x}_j - x_j)^2 + \frac{p}{2q\tau}\mathbb{E}(\hat{x}_j - \bar{x}_j)^2 + \frac{p}{q}\mathbb{E}\left(g_j(\hat{x}_j) - g_j(x_j)\right),
\end{aligned}$$

for any $x_j \in \mathbb{R}$. Then, the first conclusion of Lemma 1 is obtained by summing up the inequality above over the indices $j = 1, \ldots, p$.

In the next, we prove the second conclusion of Lemma 1. Choosing $x_i = x_i^\star$ in (A.1.2), we obtain

$$\frac{v_j x_j^\star}{n} + g_j(x_j^\star) + \frac{(x_j^\star - \bar{x}_j)^2}{2\tau} \geq \frac{v_j \tilde{x}_j}{n} + g_j(\tilde{x}_j) + \frac{(\tilde{x}_j - \bar{x}_j)^2}{2\tau} + \left(\frac{1}{2\tau} + \frac{\lambda}{2}\right)(\tilde{x}_j - x_j^\star)^2. \quad \text{(A.1.7)}$$

According to the property (5.1.6) of a saddle point, the primal optimal solution $x^\star$ satisfies $x^\star = \operatorname{argmin}_{x\in\mathbb{R}^n}\left\{\frac{1}{n}(y^\star)^T Ax + g(x)\right\}$. Due to the decomposable structure (5.1.2) of $g(x)$, each coordinate $x_j^\star$ of $x^\star$ can be solved independently. Since g_j is λ-strongly convex, the optimality of $x_j^\star$ implies

$$\frac{1}{n}\langle A^j, y^\star\rangle \tilde{x}_j + g_j(\tilde{x}_j) \geq \frac{1}{n}\langle A^j, y^\star\rangle x_j^\star + g_j(x_j^\star) + \frac{\lambda}{2}(\tilde{x}_j - x_j^\star)^2. \quad \text{(A.1.8)}$$

Summing up (A.1.7) and (A.1.8) gives us

$$\frac{1}{2\tau}(x_j^\star - \bar{x}_j)^2 \geq \left(\frac{1}{2\tau} + \lambda\right)(\tilde{x}_j - x_j^\star)^2 + \frac{1}{2\tau}(\tilde{x}_j - \bar{x}_j)^2 + \frac{1}{n}(v_j - \langle A^j, y^\star\rangle)(\tilde{x}_j - x_j^\star). \quad \text{(A.1.9)}$$

By equalities (A.1.3), (A.1.4) and (A.1.5), we can represent all the terms in (A.1.9) that involve $\tilde{x}_j$ by the terms that only contain $\hat{x}_j$, $\bar{x}_j$ and x_j. Then, we obtain

$$\begin{aligned} & \left(\frac{p}{2q\tau} + \frac{(p-q)\lambda}{q}\right)(x_j^\star - \bar{x}_j)^2 - \left(\frac{p}{2q\tau} + \frac{p\lambda}{q}\right)\mathbb{E}(\hat{x}_j - x_j^\star)^2 \\ \geq & \ \frac{p}{2q\tau}\mathbb{E}(\hat{x}_j - \bar{x}_j)^2 + \frac{1}{n}(v_j - \langle A^j, y^\star\rangle)\left(\bar{x}_j + \frac{p}{q}(\mathbb{E}\hat{x}_j - \bar{x}_j) - x_j^\star\right). \end{aligned}$$

Then, the second conclusion is obtained by summing up the inequality above over the indices $j = 1, \ldots, p$. ■

Lemma 2 *Given any $v \in \mathbb{R}^n$ and $\bar{y} \in \mathbb{R}^n$, if we uniformly and randomly choose a set of indices $I \subset \{1, 2, \ldots, n\}$ with $|I| = m$ and solve an $\hat{y} \in \mathbb{R}^n$ with*

$$\hat{y}_i = \begin{cases} \operatorname{argmax}_{\beta\in\mathbb{R}}\left\{\frac{1}{n}u_i\beta - \frac{\phi_i^*(\beta)}{n} - \frac{1}{2\sigma}(\beta - \bar{y}_i)^2\right\} & \text{if } i \in I \\ \bar{y}_i & \text{if } i \notin I, \end{cases} \quad \text{(A.1.10)}$$

then, any $y \in \mathbb{R}^n$, we have

$$\begin{aligned} & \left(\frac{n}{2m\sigma} + \frac{(n-m)\gamma}{2mn}\right)\|y - \bar{y}\|^2 + \frac{n-m}{mn}\sum_{i=1}^n\left(\phi_i^*(y_i^{(t)}) - \phi_i^*(y_i)\right) + \frac{1}{n}\mathbb{E}\left\langle u, \bar{y} + \frac{n}{m}(\hat{y} - \bar{y}) - y\right\rangle \\ \geq & \ \left(\frac{n}{2m\sigma} + \frac{\gamma}{2m}\right)\mathbb{E}\|\hat{y} - y\|^2 + \frac{n}{2m\sigma}\mathbb{E}\|\hat{y} - \bar{y}\|^2 + \frac{1}{m}\sum_{i=1}^n\mathbb{E}\left(\phi_i^*(y_i^{(t+1)}) - \phi_i^*(y_i)\right) \end{aligned}$$

and

$$\begin{aligned} & \left(\frac{n}{2m\sigma} + \frac{(n-m)\gamma}{mn}\right)\|y^\star - \bar{y}\|^2 \\ \geq & \ \left(\frac{n}{2m\sigma} + \frac{\gamma}{m}\right)\mathbb{E}\|\hat{y} - y^\star\|^2 + \frac{n}{2m\sigma}\mathbb{E}\|\hat{y} - \bar{y}\|^2 - \frac{1}{n}\mathbb{E}\left\langle u - Ax^\star, \bar{y} + \frac{n}{m}(\hat{y} - \bar{y}) - y^\star\right\rangle. \end{aligned}$$

where the expectation $\mathbb{E}$ is taken over I.

Proof: The proof is very similar to that of Lemma 1, and thus, is omitted. ■

A.2 Convergence in distance to the optimal solution

We use $\mathbb{E}_t$ to represent the expectation conditioned on $y^{(0)}, x^{(0)}, \ldots, y^{(t)}, x^{(t)}$, and $\mathbb{E}_{t+}$ the expectation conditioned on $y^{(0)}, x^{(0)}, \ldots, y^{(t)}, x^{(t)}, y^{(t+1)}$. Lemma 1 and Lemma 2 in the previous section provide the basis for the following proposition, which is the key to prove Theorem 5.4.1.

Proposition 1 *Let $x^{(t)}$, $x^{(t+1)}$, $y^{(t)}$ and $y^{(t+1)}$ generated as in Algorithm 3 for $t = 0, 1, \ldots$ with the parameters τ and σ satisfying $\tau\sigma = \frac{nmq}{4p\Lambda}$. We have*

$$\begin{aligned}
&\left(\frac{p}{2q\tau} + \frac{(p-q)\lambda}{q}\right)\|x^\star - x^{(t)}\|^2 + \left(\frac{n}{2m\sigma} + \frac{(n-m)\gamma}{mn}\right)\|y^\star - y^{(t)}\|^2 \\
&+\frac{\theta}{n}\left\langle A(x^{(t)} - x^{(t-1)}), y^{(t)} - y^\star\right\rangle + \frac{\theta\|x^{(t)} - x^{(t-1)}\|^2}{4\tau} \\
\geq\ &\left(\frac{p}{2q\tau} + \frac{p\lambda}{q}\right)\mathbb{E}_t\|x^{(t+1)} - x^\star\|^2 + \left(\frac{n}{2m\sigma} + \frac{\gamma}{m}\right)\mathbb{E}_t\|y^{(t+1)} - y^\star\|^2 \\
&+\left(\frac{p}{2q\tau} - \frac{(n-m)p}{4n\tau q}\right)\mathbb{E}_t\|x^{(t+1)} - x^{(t)}\|^2 + \left(\frac{n}{2m\sigma} - \frac{\theta nq}{4\sigma mp} - \frac{n-m}{4\sigma m}\right)\mathbb{E}_t\|y^{(t+1)} - y^{(t)}\|^2 \\
&+\frac{p}{nq}\mathbb{E}_t\left\langle A^T(y^{(t+1)} - y^\star), x^{(t+1)} - x^{(t)}\right\rangle. \qquad \text{(A.2.1)}
\end{aligned}$$

Proof: Let $x^{(t)}$, $x^{(t+1)}$ and $\bar{y}^{(t+1)}$ generated as in Algorithm 3. By the second conclusion of Lemma 1 and the tower property $\mathbb{E}_t\mathbb{E}_{t+} = \mathbb{E}_t$, we have

$$\begin{aligned}
\left(\frac{p}{2q\tau} + \frac{(p-q)\lambda}{q}\right)\|x^\star - x^{(t)}\|^2 \ \geq\ &\left(\frac{p}{2q\tau} + \frac{p\lambda}{q}\right)\mathbb{E}_t\|x^{(t+1)} - x^\star\|^2 + \frac{p}{2q\tau}\mathbb{E}_t\|x^{(t+1)} - x^{(t)}\|^2 \\
&+\frac{1}{n}\mathbb{E}_t\left\langle A^T(\bar{y}^{(t+1)} - y^\star), x^{(t)} + \frac{p}{q}(x^{(t+1)} - x^{(t)}) - x^\star\right\rangle.
\end{aligned}$$

Similarly, let $y^{(t)}$, $y^{(t+1)}$ and $\bar{x}^{(t)}$ generated as in Algorithm 3. By the second conclusion of Lemma 2, we have

$$\begin{aligned}
\left(\frac{n}{2m\sigma} + \frac{(n-m)\gamma}{mn}\right)\|y^\star - y^{(t)}\|^2 \ \geq\ &\left(\frac{n}{2m\sigma} + \frac{\gamma}{m}\right)\mathbb{E}_t\|y^{(t+1)} - y^\star\|^2 + \frac{n}{2m\sigma}\mathbb{E}_t\|y^{(t+1)} - y^{(t)}\|^2 \\
&-\frac{1}{n}\mathbb{E}_t\left\langle A(\bar{x}^{(t)} - x^\star), y^{(t)} + \frac{n}{m}(y^{(t+1)} - y^{(t)}) - y^\star\right\rangle.
\end{aligned}$$

Summing up these two inequalities, we have

$$\begin{aligned}
&\left(\frac{p}{2q\tau} + \frac{(p-q)\lambda}{q}\right)\|x^\star - x^{(t)}\|^2 + \left(\frac{n}{2m\sigma} + \frac{(n-m)\gamma}{mn}\right)\|y^\star - y^{(t)}\|^2 \\
\geq\ &\left(\frac{p}{2q\tau} + \frac{p\lambda}{q}\right)\mathbb{E}_t\|x^{(t+1)} - x^\star\|^2 + \left(\frac{n}{2m\sigma} + \frac{\gamma}{m}\right)\mathbb{E}_t\|y^{(t+1)} - y^\star\|^2 + \frac{p}{2q\tau}\mathbb{E}_t\|x^{(t+1)} - x^{(t)}\|^2 \\
&+\frac{n}{2m\sigma}\mathbb{E}_t\|y^{(t+1)} - y^{(t)}\|^2 + \frac{1}{n}\mathbb{E}_t\left\langle A^T(\bar{y}^{(t+1)} - y^\star), x^{(t)} + \frac{p}{q}(x^{(t+1)} - x^{(t)}) - x^\star\right\rangle \qquad \text{(A.2.2)} \\
&-\frac{1}{n}\mathbb{E}_t\left\langle A(\bar{x}^{(t)} - x^\star), y^{(t)} + \frac{n}{m}(y^{(t+1)} - y^{(t)}) - y^\star\right\rangle.
\end{aligned}$$

By the definition of $\bar{y}^{(t+1)}$ in Algorithm 3, we have $\bar{y}^{(t+1)} - y^\star = y^{(t+1)} - y^\star + \frac{n-m}{m}(y^{(t+1)} - y^{(t)})$, which implies

$$\begin{aligned}
&\frac{1}{n}\left\langle A^T(\bar{y}^{(t+1)} - y^\star), x^{(t)} + \frac{p}{q}(x^{(t+1)} - x^{(t)}) - x^\star \right\rangle \\
= \;& \frac{1}{n}\left\langle A^T(y^{(t+1)} - y^\star + \frac{n-m}{m}(y^{(t+1)} - y^{(t)})), x^{(t)} + \frac{p}{q}(x^{(t+1)} - x^{(t)}) - x^\star \right\rangle \\
= \;& \frac{p}{nq}\left\langle A^T(y^{(t+1)} - y^\star), x^{(t+1)} - x^{(t)} \right\rangle + \frac{1}{n}\left\langle A^T(y^{(t+1)} - y^\star), x^{(t)} - x^\star \right\rangle \qquad \text{(A.2.3)} \\
&+ \frac{(n-m)p}{nmq}\left\langle A^T(y^{(t+1)} - y^{(t)})), x^{(t+1)} - x^{(t)} \right\rangle + \frac{n-m}{nm}\left\langle A^T(y^{(t+1)} - y^{(t)})), x^{(t)} - x^\star \right\rangle.
\end{aligned}$$

Similarly, by the definition of $\bar{x}^{(t)}$ in Algorithm 3, we have $\bar{x}^{(t)} - x^\star = x^{(t)} - x^\star + \theta(x^{(t)} - x^{(t-1)})$, which implies

$$\begin{aligned}
&\frac{1}{n}\left\langle A(\bar{x}^{(t)} - x^\star), y^{(t)} + \frac{n}{m}(y^{(t+1)} - y^{(t)}) - y^\star \right\rangle \\
= \;& \frac{1}{n}\left\langle A(x^{(t)} - x^\star + \theta(x^{(t)} - x^{(t-1)})), y^{(t)} + \frac{n}{m}(y^{(t+1)} - y^{(t)}) - y^\star \right\rangle \\
= \;& \frac{1}{m}\left\langle A(x^{(t)} - x^\star), y^{(t+1)} - y^{(t)} \right\rangle + \frac{1}{n}\left\langle A(x^{(t)} - x^\star), y^{(t)} - y^\star \right\rangle \\
&+ \frac{\theta}{m}\left\langle A(x^{(t)} - x^{(t-1)}), y^{(t+1)} - y^{(t)} \right\rangle + \frac{\theta}{n}\left\langle A(x^{(t)} - x^{(t-1)}), y^{(t)} - y^\star \right\rangle. \qquad \text{(A.2.4)}
\end{aligned}$$

According to (A.2.3) and (A.2.4), the last two terms in the right hand side of (A.2.3) within conditional expectation $\mathbb{E}_t$ can be represented as

$$\begin{aligned}
&\frac{1}{n}\left\langle A^T(\bar{y}^{(t+1)} - y^\star), x^{(t)} + \frac{p}{q}(x^{(t+1)} - x^{(t)}) - x^\star \right\rangle \\
&- \frac{1}{n}\left\langle A(\bar{x}^{(t)} - x^\star), y^{(t)} + \frac{n}{m}(y^{(t+1)} - y^{(t)}) - y^\star \right\rangle \\
= \;& \frac{p}{nq}\left\langle A^T(y^{(t+1)} - y^\star), x^{(t+1)} - x^{(t)} \right\rangle + \frac{1}{n}\left\langle A^T(y^{(t+1)} - y^\star), x^{(t)} - x^\star \right\rangle \\
&+ \frac{(n-m)p}{nmq}\left\langle A^T(y^{(t+1)} - y^{(t)})), x^{(t+1)} - x^{(t)} \right\rangle + \frac{n-m}{nm}\left\langle A^T(y^{(t+1)} - y^{(t)})), x^{(t)} - x^\star \right\rangle \\
&- \frac{1}{m}\left\langle A(x^{(t)} - x^\star), y^{(t+1)} - y^{(t)} \right\rangle - \frac{1}{n}\left\langle A(x^{(t)} - x^\star), y^{(t)} - y^\star \right\rangle \\
&- \frac{\theta}{m}\left\langle A(x^{(t)} - x^{(t-1)}), y^{(t+1)} - y^{(t)} \right\rangle - \frac{\theta}{n}\left\langle A(x^{(t)} - x^{(t-1)}), y^{(t)} - y^\star \right\rangle \\
= \;& \frac{p}{nq}\left\langle A^T(y^{(t+1)} - y^\star), x^{(t+1)} - x^{(t)} \right\rangle + \frac{(n-m)p}{nmq}\left\langle A^T(y^{(t+1)} - y^{(t)})), x^{(t+1)} - x^{(t)} \right\rangle \\
&- \frac{\theta}{m}\left\langle A(x^{(t)} - x^{(t-1)}), y^{(t+1)} - y^{(t)} \right\rangle - \frac{\theta}{n}\left\langle A(x^{(t)} - x^{(t-1)}), y^{(t)} - y^\star \right\rangle. \qquad \text{(A.2.5)}
\end{aligned}$$

In the next, we establish some lower bounds for each of the four terms in (A.2.5).

Note that $x^{(t)} - x^{(t-1)}$ is a sparse vector with non-zero values only in the coordinates indexed by J. Hence, by Young's inequality, we have

$$\begin{aligned} -\left\langle A(x^{(t)} - x^{(t-1)}), y^{(t+1)} - y^{(t)} \right\rangle &\geq -\frac{\tau}{m}\|(A^J)^T(y^{(t+1)} - y^{(t)})\|^2 - \frac{\|x^{(t)} - x^{(t-1)}\|^2}{4\tau/m} \\ &\geq -\frac{\tau\Lambda}{m}\|y^{(t+1)} - y^{(t)}\|^2 - \frac{\|x^{(t)} - x^{(t-1)}\|^2}{4\tau/m} \\ &= -\frac{nq}{4\sigma p}\|y^{(t+1)} - y^{(t)}\|^2 - \frac{\|x^{(t)} - x^{(t-1)}\|^2}{4\tau/m}. \quad \text{(A.2.6)} \end{aligned}$$

Here, the second inequality is because $y^{(t+1)} - y^{(t)}$ has non-zero values only in the coordinates indexed by I so that, by the definition (5.1.8) of Λ,

$$\|(A^J)^T(y^{(t+1)} - y^{(t)})\|^2 = \|(A_I^J)^T(y_I^{(t+1)} - y_I^{(t)})\|^2 \leq \Lambda\|y_I^{(t+1)} - y_I^{(t)}\|^2 = \Lambda\|y^{(t+1)} - y^{(t)}\|^2,$$

and the last equality is because $\tau\sigma = \frac{nmq}{4p\Lambda}$.

A similar argument implies

$$\left\langle A^T(y^{(t+1)} - y^{(t)}), x^{(t+1)} - x^{(t)} \right\rangle \geq -\frac{nq}{4\sigma p}\|y^{(t+1)} - y^{(t)}\|^2 - \frac{\|x^{(t+1)} - x^{(t)}\|^2}{4\tau/m}. \quad \text{(A.2.7)}$$

Applying (A.2.6) and (A.2.7) to the right hand side of (A.2.5), we have

$$\begin{aligned} &\frac{1}{n}\left\langle A^T(\bar{y}^{(t+1)} - y^\star), x^{(t)} + \frac{p}{q}(x^{(t+1)} - x^{(t)}) - x^\star \right\rangle \\ &-\frac{1}{n}\left\langle A(\bar{x}^{(t)} - x^\star), y^{(t)} + \frac{n}{m}(y^{(t+1)} - y^{(t)}) - y^\star \right\rangle \\ \geq\ &\frac{p}{nq}\left\langle A^T(y^{(t+1)} - y^\star), x^{(t+1)} - x^{(t)} \right\rangle - \frac{\theta}{n}\left\langle A(x^{(t)} - x^{(t-1)}), y^{(t)} - y^\star \right\rangle \quad \text{(A.2.8)} \\ &-\left(\frac{\theta nq}{4\sigma mp} + \frac{n-m}{4\sigma m}\right)\|y^{(t+1)} - y^{(t)}\|^2 - \frac{\theta\|x^{(t)} - x^{(t-1)}\|^2}{4\tau} - \frac{(n-m)p\|x^{(t+1)} - x^{(t)}\|^2}{4\tau nq} \end{aligned}$$

The conclusion (A.2.1) is obtained by combining (A.2.3) and the conditional expectation of (A.2.8). ■

Based on Proposition 1, we can prove Theorem 5.4.1.

Proof: [**Theorem 5.4.1**] We first show how to derive the forms of τ and σ in (5.4.6) and (5.4.7) from the last two equations in (5.4.5). Let $Q_1 = \frac{p}{2q\lambda\tau}$ and $Q_2 = \frac{n^2}{2m\gamma\sigma}$. The last two equations in (5.4.5) imply

$$Q_1 Q_2 = \frac{pn^2}{4qm\lambda\gamma\tau\sigma} = \frac{(np)^2\Lambda}{(mq)^2 n\lambda\gamma}, \quad Q_1 + \frac{p}{q} = Q_2 + \frac{n}{m}.$$

Solving the values of Q_1 and Q_2 from these equations, we obtain

$$\frac{p}{2q\lambda\tau} = Q_1 = \frac{1}{2}\left(\frac{n}{m} - \frac{p}{q}\right) + \frac{1}{2}\sqrt{\left(\frac{n}{m} - \frac{p}{q}\right)^2 + \frac{4(np)^2\Lambda}{(mq)^2 n\lambda\gamma}}, \quad \text{(A.2.9)}$$

$$\frac{n^2}{2m\gamma\sigma} = Q_2 = \frac{1}{2}\left(\frac{p}{q} - \frac{n}{m}\right) + \frac{1}{2}\sqrt{\left(\frac{n}{m} - \frac{p}{q}\right)^2 + \frac{4(np)^2\Lambda}{(mq)^2 n\lambda\gamma}}, \quad \text{(A.2.10)}$$

from which (5.4.6) and (5.4.7) can be derived.

To derive the main conclusion of Theorem 5.4.1 from Proposition 1, we want to show that the following inequalities are satisfied by the choices for θ, τ and σ in (5.4.5).

$$\left(\frac{p}{2q\tau}+\frac{p\lambda}{q}\right)\frac{\theta q}{p} \;\geq\; \left(\frac{p}{2q\tau}+\frac{(p-q)\lambda}{q}\right), \tag{A.2.11}$$

$$\left(\frac{n}{2m\sigma}+\frac{\gamma}{m}\right)\frac{\theta q}{p} \;\geq\; \left(\frac{n}{2m\sigma}+\frac{(n-m)\gamma}{mn}\right), \tag{A.2.12}$$

$$\frac{n}{2m\sigma}-\frac{\theta nq}{4\sigma mp}-\frac{n-m}{4\sigma m} \;\geq\; 0, \tag{A.2.13}$$

$$\left(\frac{p}{2q\tau}-\frac{(n-m)p}{4n\tau q}\right)\frac{\theta q}{p} \;\geq\; \frac{\theta}{4\tau}. \tag{A.2.14}$$

In fact, (A.2.11) holds since (A.2.9) implies[1]

$$\frac{p}{2q\lambda\tau}+\frac{p}{q}\leq\frac{1}{2}\left(\frac{n}{m}+\frac{p}{q}\right)+\frac{1}{2}\left|\frac{n}{m}-\frac{p}{q}\right|+\frac{np\sqrt{\Lambda}}{mq\sqrt{n\lambda\gamma}}=\max\left\{\frac{p}{q},\frac{n}{m}\right\}+\frac{np\sqrt{\Lambda}}{mq\sqrt{n\lambda\gamma}}$$

so that

$$\left(\frac{p}{2q\tau}+\frac{(p-q)\lambda}{q}\right)/\left(\frac{p}{2q\tau}+\frac{p\lambda}{q}\right)=1-\frac{1}{\frac{p}{2q\lambda\tau}+\frac{p}{q}}\leq 1-\frac{1}{\max\left\{\frac{p}{q},\frac{n}{m}\right\}+\frac{np\sqrt{\Lambda}}{mq\sqrt{n\lambda\gamma}}}=\frac{\theta q}{p}.$$

Similarly, (A.2.12) holds since (A.2.10) implies

$$\frac{n^2}{2m\gamma\sigma}+\frac{n}{m}\leq\frac{1}{2}\left(\frac{n}{m}+\frac{p}{q}\right)+\frac{1}{2}\left|\frac{n}{m}-\frac{p}{q}\right|+\frac{np\sqrt{\Lambda}}{mq\sqrt{n\lambda\gamma}}=\max\left\{\frac{p}{q},\frac{n}{m}\right\}+\frac{np\sqrt{\Lambda}}{mq\sqrt{n\lambda\gamma}}$$

so that

$$\left(\frac{n}{2m\sigma}+\frac{(n-m)\gamma}{mn}\right)/\left(\frac{n}{2m\sigma}+\frac{\gamma}{m}\right)=1-\frac{1}{\frac{n^2}{2m\gamma\sigma}+\frac{n}{m}}\leq 1-\frac{1}{\max\left\{\frac{p}{q},\frac{n}{m}\right\}+\frac{np\sqrt{\Lambda}}{mq\sqrt{n\lambda\gamma}}}=\frac{\theta q}{p}.$$

The inequality (A.2.13) holds because $\frac{n}{2m\sigma}-\frac{\theta nq}{4\sigma mp}-\frac{n-m}{4\sigma m}\geq\frac{n}{2m\sigma}-\frac{n}{4\sigma m}-\frac{n}{4\sigma m}=0$, where we use the fact that $\frac{\theta q}{p}\leq 1$. The inequality (A.2.14) holds because $\left(\frac{p}{2q\tau}-\frac{(n-m)p}{4n\tau q}\right)\frac{\theta q}{p}=\theta\left(\frac{1}{2\tau}-\frac{(n-m)}{4n\tau}\right)\geq\theta\left(\frac{1}{2\tau}-\frac{1}{4\tau}\right)=\frac{\theta}{4\tau}$.

Applying the four inequalities (A.2.11), (A.2.12), (A.2.13) and (A.2.14) to the coefficients of (A.2.1) from Proposition 1 leads to $\mathbb{E}_t\Delta^{(t+1)}\leq\left(\frac{\theta q}{p}\right)\Delta^{(t)}$ for any $t\geq 0$, where

$$\begin{aligned}\Delta^{(t)} \;=\; &\left(\frac{p}{2q\tau}+\frac{p\lambda}{q}\right)\|x^\star-x^{(t)}\|^2+\left(\frac{n}{2m\sigma}+\frac{\gamma}{m}\right)\|y^\star-y^{(t)}\|^2\\ &+\frac{p}{qn}\left\langle A(x^{(t)}-x^{(t-1)}),y^{(t)}-y^\star\right\rangle+\left(\frac{p}{2q\tau}-\frac{(n-m)p}{4n\tau q}\right)\|x^{(t)}-x^{(t-1)}\|^2\end{aligned} \tag{A.2.15}$$

[1]Here and when we show (A.2.12), we use the simple fact that $\sqrt{a^2+b^2}\leq a+b$ when $a\geq 0$ and $b\geq 0$.

Applying this result recursively gives $\mathbb{E}\Delta^{(t)} \leq \left(\frac{\theta q}{p}\right)^t \Delta^{(0)}$ where

$$\Delta^{(0)} = \left(\frac{p}{2q\tau} + \frac{p\lambda}{q}\right)\|x^\star - x^{(0)}\|^2 + \left(\frac{n}{2m\sigma} + \frac{\gamma}{m}\right)\|y^\star - y^{(0)}\|^2$$

because $(x^{(0)}, y^{(0)}) = (x^{(-1)}, y^{(-1)})$.

Let $\tilde{I}$ be a uniformly random subset of $\{1, 2, \ldots, n\}$ with $|\tilde{I}| = m$, i.e., each index in $\{1, 2, \ldots, n\}$ is contained in $\tilde{I}$ with a probability of $\frac{m}{n}$. By Jensen's inequality and (5.1.8), we have

$$\frac{m^2}{n^2}\|(A^J)^T(y^{(t)} - y^\star)\|^2 = \|\mathbb{E}(A^J_{\tilde{I}})^T(y^{(t)}_{\tilde{I}} - y^\star_{\tilde{I}})\|^2 \leq \mathbb{E}\|(A^J_{\tilde{I}})^T(y^{(t)}_{\tilde{I}} - y^\star_{\tilde{I}})\|^2 = \frac{m\Lambda}{n}\|y^{(t)} - y^\star\|^2,$$

where the expectation $\mathbb{E}$ is taken over $\tilde{I}$. This result further implies

$$\|(A^J)^T(y^{(t)} - y^\star)\|^2 \leq \frac{n\Lambda}{m}\|y^{(t)} - y^\star\|^2. \tag{A.2.16}$$

Note that $x^{(t)} - x^{(t-1)}$ is a sparse vector with non-zero values only in the coordinates indexed by J. Hence, by Young's inequality, we have

$$\begin{aligned}
\left\langle A(x^{(t)} - x^{(t-1)}), y^{(t)} - y^\star \right\rangle &\geq -\frac{\tau}{n}\|(A^J)^T(y^{(t)} - y^\star)\|^2 - \frac{\|x^{(t)} - x^{(t-1)}\|^2}{4\tau/n} \\
&\geq -\frac{\tau\Lambda}{m}\|y^{(t)} - y^\star\|^2 - \frac{\|x^{(t)} - x^{(t-1)}\|^2}{4\tau/n} \\
&= -\frac{nq}{4\sigma p}\|y^{(t)} - y^\star\|^2 - \frac{\|x^{(t)} - x^{(t-1)}\|^2}{4\tau/n}, \qquad \text{(A.2.17)}
\end{aligned}$$

where the second inequality is because of (A.2.16) and the last equality is because $\tau\sigma = \frac{nmq}{4\Lambda p}$. Applying (A.2.17) to the right hand side of (A.2.15) leads to

$$\begin{aligned}
\Delta^{(t)} &\geq \left(\frac{p}{2q\tau} + \frac{p\lambda}{q}\right)\|x^\star - x^{(t)}\|^2 + \left(\frac{n}{2m\sigma} + \frac{\gamma}{m} - \frac{1}{4\sigma}\right)\|y^\star - y^{(t)}\|^2 \\
&\quad + \left(\frac{p}{2q\tau} - \frac{(n-m)p}{4n\tau q} - \frac{p}{4\tau q}\right)\|x^{(t)} - x^{(t-1)}\|^2 \\
&\geq \left(\frac{p}{2q\tau} + \frac{p\lambda}{q}\right)\|x^\star - x^{(t)}\|^2 + \left(\frac{n}{4m\sigma} + \frac{\gamma}{m}\right)\|y^\star - y^{(t)}\|^2,
\end{aligned}$$

where the second inequalities holds because $\frac{n}{2m\sigma} - \frac{1}{4\sigma} \geq \frac{n}{2m\sigma} - \frac{n}{4m\sigma} = \frac{n}{4m\sigma}$ and $\frac{p}{2q\tau} - \frac{(n-m)p}{4n\tau q} - \frac{p}{4\tau q} \geq \frac{p}{2q\tau} - \frac{p}{4\tau q} - \frac{p}{4\tau q} = 0$. Then, the conclusion of Theorem 5.4.1 can be obtained as

$$\begin{aligned}
&\left(\frac{p}{2q\tau} + \frac{p\lambda}{q}\right)\mathbb{E}\|x^\star - x^{(t)}\|^2 + \left(\frac{n}{4m\sigma} + \frac{\gamma}{m}\right)\mathbb{E}\|y^\star - y^{(t)}\|^2 \leq \mathbb{E}\Delta^{(t)} \leq \left(\frac{\theta q}{p}\right)^t \Delta^{(0)} \\
\leq &\left(1 - \frac{1}{\max\left\{\frac{p}{q}, \frac{n}{m}\right\} + \frac{pR\sqrt{n}}{q\sqrt{m\lambda\gamma}}}\right)^t \left[\left(\frac{p}{2q\tau} + \frac{p\lambda}{q}\right)\|x^\star - x^{(0)}\|^2 + \left(\frac{n}{2m\sigma} + \frac{\gamma}{m}\right)\|y^\star - y^{(0)}\|^2\right].
\end{aligned}$$

■

A.3 Convergence of objective gap

To establish the convergence of primal-dual gap (Theorem 5.4.2), we define the following two functions

$$\tilde{P}(x) \equiv g(x) + \frac{1}{n}(y^\star)^T Ax - \left[g(x^\star) + \frac{1}{n}(y^\star)^T Ax^\star \right], \tag{A.3.1}$$

$$\tilde{D}(y) \equiv \frac{1}{n} y^T Ax^\star - \frac{1}{n}\sum_{i=1}^n \phi_i^*(y_i) - \left[\frac{1}{n}(y^\star)^T Ax^\star - \frac{1}{n}\sum_{i=1}^n \phi_i^*(y_i^\star) \right]. \tag{A.3.2}$$

Note that

$$\tilde{P}(x) \geq \frac{\lambda}{2}\|x - x^\star\|^2 \text{ and } \tilde{D}(y) \leq -\frac{\gamma}{2n}\|y - y^\star\|^2 \tag{A.3.3}$$

and

$$\tilde{P}(x) - \tilde{D}(y) \leq P(x) - D(y) \tag{A.3.4}$$

for any $x \in \mathbb{R}^p$ and any $y \in \mathbb{R}^n$ because of (5.1.6) and the strong convexity of $g(x)$ and $\frac{1}{n}\sum_{i=1}^n \phi_i^*(y_i)$. Then, we provide the following proposition, which is the key to prove Theorem 5.4.2.

Proposition 2 *Let $x^{(t)}$, $x^{(t+1)}$, $y^{(t)}$ and $y^{(t+1)}$ generated as in Algorithm 3 for $t = 0, 1, \ldots$ with the parameters τ and σ satisfying $\tau\sigma = \frac{nmq}{4p\Lambda}$. We have*

$$\begin{aligned}
&\left(\frac{p}{2q\tau} + \frac{(p-q)\lambda}{2q}\right)\|x^\star - x^{(t)}\|^2 + \left(\frac{n}{2m\sigma} + \frac{(n-m)\gamma}{2mn}\right)\|y^\star - y^{(t)}\|^2 \\
&\frac{\theta}{n}\left\langle A(x^{(t)} - x^{(t-1)}), y^{(t)} - y^\star \right\rangle + \frac{\theta\|x^{(t)} - x^{(t-1)}\|^2}{4\tau} + \frac{p-q}{q}\tilde{P}(x^{(t)}) - \frac{n-m}{m}\tilde{D}(y^{(t)}) \\
\geq\ &\left(\frac{p}{2q\tau} + \frac{p\lambda}{2q}\right)\mathbb{E}_t\|x^{(t+1)} - x^\star\|^2 + \left(\frac{n}{2m\sigma} + \frac{\gamma}{2m}\right)\mathbb{E}_t\|y^{(t+1)} - y^\star\|^2 \\
&+ \left(\frac{p}{2q\tau} - \frac{(n-m)p}{4n\tau q}\right)\mathbb{E}_t\|x^{(t+1)} - x^{(t)}\|^2 + \left(\frac{n}{2m\sigma} - \frac{\theta nq}{4\sigma mp} - \frac{n-m}{4\sigma m}\right)\mathbb{E}_t\|y^{(t+1)} - y^{(t)}\|^2 \\
&+ \frac{p}{nq}\mathbb{E}_t\left\langle A^T(y^{(t+1)} - y^\star), x^{(t+1)} - x^{(t)} \right\rangle + \frac{p}{q}\mathbb{E}_t\tilde{P}(x^{(t+1)}) - \frac{n}{m}\mathbb{E}_t\tilde{D}(y^{(t+1)}).
\end{aligned} \tag{A.3.5}$$

Proof: Let $x^{(t)}$, $x^{(t+1)}$ and $\bar{y}^{(t+1)}$ generated as in Algorithm 3. By the first conclusion of Lemma 1 and the tower property $\mathbb{E}_t\mathbb{E}_{t+} = \mathbb{E}_t$, for any $x \in \mathbb{R}^p$,

$$\begin{aligned}
&\left(\frac{p}{2q\tau} + \frac{(p-q)\lambda}{2q}\right)\|x - x^{(t)}\|^2 + \frac{p-q}{q}\left(g(x^{(t)}) - g(x)\right) \\
\geq\ &\left(\frac{p}{2q\tau} + \frac{p\lambda}{2q}\right)\mathbb{E}\|x^{(t+1)} - x\|^2 + \frac{p}{2q\tau}\mathbb{E}\|x^{(t+1)} - x^{(t)}\|^2 + \frac{p}{q}\mathbb{E}\left(g(x^{(t+1)}) - g(x)\right) \\
&+ \frac{1}{n}\mathbb{E}\left\langle A^T\bar{y}^{(t+1)}, x^{(t)} + \frac{p}{q}(x^{(t+1)} - x^{(t)}) - x \right\rangle.
\end{aligned} \tag{A.3.6}$$

Let $y^{(t)}$, $y^{(t+1)}$ and $\bar{x}^{(t)}$ generated as in Algorithm 3. By the first conclusion of Lemma 2, we have, for any $y \in \mathbb{R}^n$,

$$\begin{aligned}
&\left(\frac{n}{2m\sigma}+\frac{(n-m)\gamma}{2mn}\right)\|y-y^{(t)}\|^2+\frac{n-m}{mn}\sum_{i=1}^{n}\left(\phi_i^*(y_i^{(t)})-\phi_i^*(y_i)\right) \qquad \text{(A.3.7)}\\
\geq\ &\left(\frac{n}{2m\sigma}+\frac{\gamma}{2m}\right)\mathbb{E}\|y^{(t+1)}-y\|^2+\frac{n}{2m\sigma}\mathbb{E}\|y^{(t+1)}-y^{(t)}\|^2+\frac{1}{m}\sum_{i=1}^{n}\mathbb{E}\left(\phi_i^*(y_i^{(t+1)})-\phi_i^*(y_i)\right)\\
&-\frac{1}{n}\mathbb{E}\left\langle A\bar{x}^{(t)},y^{(t)}+\frac{n}{m}(y^{(t+1)}-y^{(t)})-y\right\rangle
\end{aligned}$$

Summing up the inequalities (A.3.6) and (A.3.7) and setting $(x,y)=(x^\star,y^\star)$ yield

$$\begin{aligned}
&\left(\frac{p}{2q\tau}+\frac{(p-q)\lambda}{2q}\right)\|x^\star-x^{(t)}\|^2+\left(\frac{n}{2m\sigma}+\frac{(n-m)\gamma}{2mn}\right)\|y^\star-y^{(t)}\|^2\\
&+\frac{p-q}{q}\left(g(x^{(t)})-g(x^\star)\right)+\frac{n-m}{mn}\sum_{i=1}^{n}\left(\phi_i^*(y_i^{(t)})-\phi_i^*(y_i^\star)\right) \qquad \text{(A.3.8)}\\
\geq\ &\left(\frac{p}{2q\tau}+\frac{p\lambda}{2q}\right)\mathbb{E}_t\|x^{(t+1)}-x^\star\|^2+\frac{p}{2q\tau}\mathbb{E}_t\|x^{(t+1)}-x^{(t)}\|^2+\left(\frac{n}{2m\sigma}+\frac{\gamma}{2m}\right)\mathbb{E}_t\|y^{(t+1)}-y^\star\|^2\\
&+\frac{p}{q}\mathbb{E}_t\left(g(x^{(t+1)})-g(x^\star)\right)+\frac{1}{m}\sum_{i=1}^{n}\mathbb{E}_t\left(\phi_i^*(y_i^{(t+1)})-\phi_i^*(y_i^\star)\right)+\frac{n}{2m\sigma}\mathbb{E}_t\|y^{(t+1)}-y^{(t)}\|^2\\
&+\frac{1}{n}\mathbb{E}_t\left\langle A^T\bar{y}^{(t+1)},x^{(t)}+\frac{p}{q}(x^{(t+1)}-x^{(t)})-x^\star\right\rangle-\frac{1}{n}\mathbb{E}_t\left\langle A\bar{x}^{(t)},y^{(t)}+\frac{n}{m}(y^{(t+1)}-y^{(t)})-y^\star\right\rangle.
\end{aligned}$$

By the definitions of $\tilde{P}(x^{(t)})$, $\tilde{D}(y^{(t)})$, $\tilde{P}(x^{(t+1)})$, and $\tilde{D}(y^{(t+1)})$, (A.3.8) is equivalent to

$$\begin{aligned}
&\left(\frac{p}{2q\tau}+\frac{(p-q)\lambda}{2q}\right)\|x^\star-x^{(t)}\|^2+\left(\frac{n}{2m\sigma}+\frac{(n-m)\gamma}{2mn}\right)\|y^\star-y^{(t)}\|^2\\
&+\frac{p-q}{q}\tilde{P}(x^{(t)})+\frac{n-m}{m}\tilde{D}(y^{(t)})\\
\geq\ &\left(\frac{p}{2q\tau}+\frac{p\lambda}{2q}\right)\mathbb{E}_t\|x^{(t+1)}-x^\star\|^2+\frac{p}{2q\tau}\mathbb{E}_t\|x^{(t+1)}-x^{(t)}\|^2+\left(\frac{n}{2m\sigma}+\frac{\gamma}{2m}\right)\mathbb{E}_t\|y^{(t+1)}-y^\star\|^2\\
&+\frac{p}{q}\mathbb{E}_t\tilde{P}(x^{(t+1)})+\frac{n}{m}\mathbb{E}_t\tilde{D}(y^{(t+1)})-\frac{1}{n}\mathbb{E}_t\left\langle A(\bar{x}^{(t)}-x^\star),y^{(t)}+\frac{n}{m}(y^{(t+1)}-y^{(t)})-y^\star\right\rangle \qquad \text{(A.3.9)}\\
&+\frac{1}{n}\mathbb{E}_t\left\langle A^T(\bar{y}^{(t+1)}-y^\star),x^{(t)}+\frac{p}{q}(x^{(t+1)}-x^{(t)})-x^\star\right\rangle+\frac{n}{2m\sigma}\mathbb{E}_t\|y^{(t+1)}-y^{(t)}\|^2.
\end{aligned}$$

which, together with (A.2.8), implies

$$
\begin{aligned}
&\left(\frac{p}{2q\tau}+\frac{(p-q)\lambda}{2q}\right)\|x^\star-x^{(t)}\|^2+\left(\frac{n}{2m\sigma}+\frac{(n-m)\gamma}{2mn}\right)\|y^\star-y^{(t)}\|^2\\
&+\frac{p-q}{q}\tilde{P}(x^{(t)})-\frac{n-m}{m}\tilde{D}(y^{(t)})\\
\geq\ &\left(\frac{p}{2q\tau}+\frac{p\lambda}{2q}\right)\mathbb{E}_t\|x^{(t+1)}-x^\star\|^2+\frac{p}{2q\tau}\mathbb{E}_t\|x^{(t+1)}-x^{(t)}\|^2+\frac{p}{q}\mathbb{E}_t\tilde{P}(x^{(t+1)})-\frac{n}{m}\mathbb{E}_t\tilde{D}(y^{(t+1)})\\
&+\left(\frac{n}{2m\sigma}+\frac{\gamma}{2m}\right)\mathbb{E}_t\|y^{(t+1)}-y^\star\|^2+\frac{n}{2m\sigma}\mathbb{E}_t\|y^{(t+1)}-y^{(t)}\|^2\\
&+\frac{p}{nq}\left\langle A^T(y^{(t+1)}-y^\star),x^{(t+1)}-x^{(t)}\right\rangle-\frac{\theta}{n}\left\langle A(x^{(t)}-x^{(t-1)}),y^{(t)}-y^\star\right\rangle\\
&-\left(\frac{\theta nq}{4\sigma mp}+\frac{n-m}{4\sigma m}\right)\|y^{(t+1)}-y^{(t)}\|^2-\frac{\theta\|x^{(t)}-x^{(t-1)}\|^2}{4\tau}-\frac{(n-m)p\|x^{(t+1)}-x^{(t)}\|^2}{4\tau nq}.
\end{aligned}
$$

The conclusion of the proposition is obtained by organizing the terms of the inequality above. ■

Based on Proposition 2, we now can prove Theorem 5.4.2.

Proof: [**Theorem 5.4.2**] We first show that the following inequalities are satisfied according to the choice for θ in (5.4.9) and the choices for τ and σ in (5.4.5).

$$
\begin{aligned}
\left(\frac{p}{2q\tau}+\frac{p\lambda}{2q}\right)\frac{\theta q}{p} &\geq \left(\frac{p}{2q\tau}+\frac{(p-q)\lambda}{2q}\right), && \text{(A.3.10)}\\
\left(\frac{n}{2m\sigma}+\frac{\gamma}{2m}\right)\frac{\theta q}{p} &\geq \left(\frac{n}{2m\sigma}+\frac{(n-m)\gamma}{2mn}\right), && \text{(A.3.11)}\\
\frac{n}{2m\sigma}-\frac{\theta nq}{4\sigma mp}-\frac{n-m}{4\sigma m} &\geq 0, && \text{(A.3.12)}\\
\left(\frac{p}{2q\tau}-\frac{(n-m)p}{4n\tau q}\right)\frac{\theta q}{p} &\geq \frac{\theta}{4\tau}, && \text{(A.3.13)}\\
\frac{\theta q}{p} &\geq \frac{p-q}{p}, && \text{(A.3.14)}\\
\frac{\theta q}{p} &\geq \frac{n-m}{n}. && \text{(A.3.15)}
\end{aligned}
$$

Since τ and σ still satisfy (5.4.5) as in Theorem 5.4.1, (A.2.9) and (A.2.10) are still satisfied. Therefore, we have

$$
\frac{p}{q\lambda\tau}+\frac{p}{q}\leq\frac{n}{m}+\left|\frac{n}{m}-\frac{p}{q}\right|+\frac{2np\sqrt{\Lambda}}{mq\sqrt{n\lambda\gamma}}=2\max\left\{\frac{p}{q},\frac{n}{m}\right\}+\frac{2np\sqrt{\Lambda}}{mq\sqrt{n\lambda\gamma}}
$$

according to (A.2.9) so that, by the new choice for θ in (5.4.9),

$$
\left(\frac{p}{2q\tau}+\frac{(p-q)\lambda}{2q}\right)/\left(\frac{p}{2q\tau}+\frac{p\lambda}{2q}\right)=1-\frac{1}{\frac{p}{q\lambda\tau}+\frac{p}{q}}\leq 1-\frac{1}{2\max\left\{\frac{p}{q},\frac{n}{m}\right\}+\frac{2np\sqrt{\Lambda}}{mq\sqrt{n\lambda\gamma}}}=\frac{\theta q}{p}.
$$

Similarly, according to (A.2.10), we have

$$\frac{n^2}{m\gamma\sigma}+\frac{n}{m}\leq\frac{p}{q}+\left|\frac{n}{m}-\frac{p}{q}\right|+\frac{2np\sqrt{\Lambda}}{mq\sqrt{n\lambda\gamma}}=2\max\left\{\frac{p}{q},\frac{n}{m}\right\}+\frac{2np\sqrt{\Lambda}}{mq\sqrt{n\lambda\gamma}}$$

so that

$$\left(\frac{n}{2m\sigma}+\frac{(n-m)\gamma}{2mn}\right)/\left(\frac{n}{2m\sigma}+\frac{\gamma}{2m}\right)=1-\frac{1}{\frac{n^2}{m\gamma\sigma}+\frac{n}{m}}\leq 1-\frac{1}{2\max\left\{\frac{p}{q},\frac{n}{m}\right\}+\frac{2np\sqrt{\Lambda}}{mq\sqrt{n\lambda\gamma}}}=\frac{\theta q}{p}.$$

Therefore, we have shown that (A.3.10) and (A.3.11) are satisfied. The inequality (A.3.12) holds because

$$\frac{n}{2m\sigma}-\frac{\theta nq}{4\sigma mp}-\frac{n-m}{4\sigma m}\geq\frac{n}{2m\sigma}-\frac{n}{4\sigma m}-\frac{n}{4\sigma m}=0,$$

where we use the fact that $\frac{\theta q}{p}\leq 1$. The inequality (A.3.13) holds because

$$\left(\frac{p}{2q\tau}-\frac{(n-m)p}{4n\tau q}\right)\frac{\theta q}{p}=\theta\left(\frac{1}{2\tau}-\frac{(n-m)}{4n\tau}\right)\geq\theta\left(\frac{1}{2\tau}-\frac{1}{4\tau}\right)=\frac{\theta}{4\tau}.$$

The inequalities (A.3.14) and (A.3.15) hold because

$$\max\left\{\frac{p-q}{p},\frac{n-m}{n}\right\}\leq 1-\frac{1}{2\max\left\{\frac{p}{q},\frac{n}{m}\right\}+\frac{2np\sqrt{\Lambda}}{mq\sqrt{n\lambda\gamma}}}=\frac{\theta q}{p}.$$

Recall that $\tilde{P}(x)\geq 0$ and $\tilde{D}(y)\leq 0$ for any $x\in\mathbb{R}^p$ and any $y\in\mathbb{R}^n$. Therefore, applying the six inequalities (A.3.10), (A.3.11), (A.3.12), (A.3.13), (A.3.14) and (A.3.15) to the coefficients of (A.3.5) from Proposition 2 leads to $\mathbb{E}_t\Delta^{(t+1)}\leq\left(\frac{\theta q}{p}\right)\Delta^{(t)}$ for any $t\geq 0$, where

$$\begin{aligned}\Delta^{(t)} &= \left(\frac{p}{2q\tau}+\frac{p\lambda}{2q}\right)\|x^{(t)}-x^\star\|^2+\left(\frac{n}{2m\sigma}+\frac{\gamma}{2m}\right)\|y^{(t)}-y^\star\|^2+\frac{p}{q}\tilde{P}(x^{(t)})-\frac{n}{m}\tilde{D}(y^{(t)})\\ &\quad+\left(\frac{p}{2q\tau}-\frac{(n-m)p}{4n\tau q}\right)\|x^{(t)}-x^{(t-1)}\|^2+\frac{p}{nq}\left\langle A^T(y^{(t)}-y^\star),x^{(t)}-x^{(t-1)}\right\rangle \quad\text{(A.3.16)}\end{aligned}$$

Applying this result recursively gives $\mathbb{E}\Delta^{(t)}\leq\left(\frac{\theta q}{p}\right)^t\Delta^{(0)}$, where

$$\begin{aligned}\Delta^{(0)} &= \left(\frac{p}{2q\tau}+\frac{p\lambda}{2q}\right)\|x^{(0)}-x^\star\|^2+\left(\frac{n}{2m\sigma}+\frac{\gamma}{2m}\right)\|y^{(0)}-y^\star\|^2 \qquad \text{(A.3.17)}\\ &\quad+\frac{p}{q}\tilde{P}(x^{(0)})-\frac{n}{m}\tilde{D}(y^{(0)})\end{aligned}$$

because $(x^{(0)},y^{(0)})=(x^{(-1)},y^{(-1)})$. Applying (A.2.17) to the right hand side of (A.3.16) leads to

$$\begin{aligned}\Delta^{(t)} &\geq \left(\frac{p}{2q\tau}+\frac{p\lambda}{2q}\right)\|x^\star-x^{(t)}\|^2+\left(\frac{n}{2m\sigma}+\frac{\gamma}{2m}-\frac{1}{4\sigma}\right)\|y^\star-y^{(t)}\|^2\\ &\quad+\left(\frac{p}{2q\tau}-\frac{(n-m)p}{4n\tau q}-\frac{p}{4\tau q}\right)\|x^{(t)}-x^{(t-1)}\|^2+\frac{p}{q}\tilde{P}(x^{(t)})-\frac{n}{m}\tilde{D}(y^{(t)})\\ &\geq \frac{p}{q}\tilde{P}(x^{(t)})-\frac{n}{m}\tilde{D}(y^{(t)}). \qquad \text{(A.3.18)}\end{aligned}$$

Combining (A.3.17), $\mathbb{E}\Delta^{(t)} \leq \left(\frac{\theta q}{p}\right)^t \Delta^{(0)}$ and (A.3.18) together, we obtain

$$\min\left\{\frac{p}{q}, \frac{n}{m}\right\} \mathbb{E}\left(\tilde{P}(x^{(t)}) - \tilde{D}(y^{(t)})\right) \leq \frac{p\mathbb{E}\tilde{P}(x^{(t)})}{q} - \frac{n\mathbb{E}\tilde{D}(y^{(t)})}{m} \leq \mathbb{E}\Delta^{(t)} \leq \left(\frac{\theta q}{p}\right)^t \Delta^{(0)} \quad \text{(A.3.19)}$$

In the next, we will establish the relationship between $\tilde{P}(x^{(t)}) - \tilde{D}(y^{(t)})$ and the actual primal-dual objective gap $P(x^{(t)}) - D(y^{(t)})$.

Because $\frac{1}{n}\sum_{i=1}^n \phi_i^*(y_i)$ is a $\frac{\gamma}{n}$-strong convex function of y, according to Theorem 1 in [130], the function defined as

$$\hat{P}(x) \equiv \max_{y\in\mathbb{R}^n}\left\{\frac{1}{n}y^T Ax - \frac{1}{n}\sum_{i=1}^n \phi_i^*(y_i)\right\} \quad \text{(A.3.20)}$$

is a convex and smooth function of x. Moreover, its gradient $\nabla\hat{P}(x)$ is Lipschitz continuous with a Lipschitz constant of $\frac{n\|A\|^2}{n^2\gamma} = \frac{\|A\|^2}{n\gamma}$ and $\nabla\hat{P}(x^\star) = \frac{1}{n}A^T y^\star$. As a result, we have

$$\hat{P}(x^{(t)}) \leq \hat{P}(x^\star) + \left\langle \nabla\hat{P}(x^\star), x^{(t)} - x^\star\right\rangle + \frac{\|A\|^2}{2n\gamma}\|x^{(t)} - x^\star\|^2 \quad \text{(A.3.21)}$$

According to the definition of the primal and dual objective functions (5.1.1) and (5.1.4) and their relationship with the saddle-point problem (5.1.5), we have

$$\begin{aligned}
& P(x^{(t)}) = \max_{y\in\mathbb{R}^n}\left\{g(x^{(t)}) + \frac{1}{n}y^T Ax^{(t)} - \frac{1}{n}\sum_{i=1}^n \phi_i^*(y_i)\right\} = g(x^{(t)}) + \hat{P}(x^{(t)}) \\
\leq\ & g(x^{(t)}) + \max_{y\in\mathbb{R}^n}\left\{\frac{1}{n}y^T Ax^\star - \frac{1}{n}\sum_{i=1}^n \phi_i^*(y_i)\right\} + \frac{1}{n}(y^\star)^T A(x^{(t)} - x^\star) + \frac{\|A\|^2}{2n\gamma}\|x^{(t)} - x^\star\|^2 \\
=\ & g(x^{(t)}) - \frac{1}{n}\sum_{i=1}^n \phi_i^*(y_i^\star) + \frac{1}{n}(y^\star)^T Ax^{(t)} + \frac{\|A\|^2}{2n\gamma}\|x^{(t)} - x^\star\|^2, \quad \text{(A.3.22)}
\end{aligned}$$

where the inequality is due to (A.3.20) and (A.3.21) and the last equality is due to (5.1.6).

Similarly, because $g(x)$ is a λ-strong convex function of x, according to Theorem 1 in [130] again, the function defined as

$$\hat{D}(y) \equiv \min_{x\in\mathbb{R}^p}\left\{\frac{1}{n}y^T Ax + g(x)\right\} \quad \text{(A.3.23)}$$

is a concave and smooth function of y. Moreover, its gradient $\nabla\hat{D}(x)$ is Lipschitz continuous with a Lipschitz constant of $\frac{\|A\|^2}{n^2\lambda}$ and $\nabla\hat{D}(y^\star) = \frac{1}{n}Ax^\star$. As a result, we have

$$\hat{D}(y^{(t)}) \geq \hat{D}(y^\star) + \left\langle \nabla\hat{D}(y^\star), y^{(t)} - y^\star\right\rangle - \frac{\|A\|^2}{2n^2\gamma}\|y^{(t)} - y^\star\|^2 \quad \text{(A.3.24)}$$

With a derivation similar to (A.3.22), we can show

$$D(y^{(t)}) = \min_{x\in\mathbb{R}^p}\left\{g(x) + \frac{1}{n}(y^{(t)})^T Ax - \frac{1}{n}\sum_{i=1}^n \phi_i^*(y_i^{(t)})\right\} = \hat{D}(y^{(t)}) - \frac{1}{n}\sum_{i=1}^n \phi_i^*(y_i^{(t)})$$

$$\begin{aligned}
\geq\ & -\frac{1}{n}\sum_{i=1}^n \phi_i^*(y_i^{(t)}) + \min_{x\in\mathbb{R}^p}\left\{g(x) + \frac{1}{n}(y^\star)^T Ax\right\} + \frac{1}{n}(y^{(t)} - y^\star)^T Ax^\star - \frac{\|A\|^2}{2\lambda n^2}\|y^{(t)} - y^\star\|^2 \\
=\ & -\frac{1}{n}\sum_{i=1}^n \phi_i^*(y_i^{(t)}) + g(x^\star) + \frac{1}{n}(y^{(t)})^T Ax^\star - \frac{\|A\|^2}{2\lambda n^2}\|y^{(t)} - y^\star\|^2. \qquad \text{(A.3.25)}
\end{aligned}$$

Combining (A.3.22) and (A.3.25) and using the definitions (A.3.1) and (A.3.2), we obtain

$$P(x^{(t)}) - D(y^{(t)}) - \frac{\|A\|^2}{2n\gamma}\|x^{(t)} - x^\star\|^2 - \frac{\|A\|^2}{2\lambda n^2}\|y^{(t)} - y^\star\|^2 \leq \tilde{P}(x^{(t)}) - \tilde{D}(y^{(t)}). \qquad \text{(A.3.26)}$$

Applying (A.3.26) to the left hand side of (A.3.19), we can show

$$\begin{aligned}
& \min\left\{\frac{p}{q}, \frac{n}{m}\right\}\mathbb{E}\left(P(x^{(t)}) - D(y^{(t)})\right) \qquad \text{(A.3.27)} \\
\leq\ & \left(\frac{\theta q}{p}\right)^t \Delta^{(0)} + \min\left\{\frac{p}{q}, \frac{n}{m}\right\}\mathbb{E}\left(\frac{\|A\|^2}{2n\gamma}\|x^{(t)} - x^\star\|^2 + \frac{\|A\|^2}{2\lambda n^2}\|y^{(t)} - y^\star\|^2\right)
\end{aligned}$$

The property (A.3.3) of $\tilde{P}$ and $\tilde{D}$ and (A.3.18) imply

$$\begin{aligned}
& \frac{\lambda p\mathbb{E}\|x^\star - x^{(t)}\|^2}{2q} + \frac{\gamma\mathbb{E}\|y^\star - y^{(t)}\|^2}{2m} \\
\leq\ & \frac{p\mathbb{E}\tilde{P}(x^{(t)})}{q} - \frac{n\mathbb{E}\tilde{D}(y^{(t)})}{m} \leq \mathbb{E}\Delta^{(t)} \leq \left(\frac{\theta q}{p}\right)^t \Delta^{(0)} \qquad \text{(A.3.28)}
\end{aligned}$$

which, together with (A.3.27), further implies

$$\begin{aligned}
& \min\left\{\frac{p}{q}, \frac{n}{m}\right\}\mathbb{E}\left(P(x^{(t)}) - D(y^{(t)})\right) \\
\leq\ & \left\{1 + \frac{\min\left\{\frac{p}{q}, \frac{n}{m}\right\}\max\left\{\frac{\|A\|^2}{n\gamma}, \frac{\|A\|^2}{\lambda n^2}\right\}}{\min\left\{\frac{\lambda p}{q}, \frac{\gamma}{m}\right\}}\right\}\left(\frac{\theta q}{p}\right)^t \Delta^{(0)}. \qquad \text{(A.3.29)}
\end{aligned}$$

It is from (A.3.4) that

$$\frac{p}{q}\tilde{P}(x^{(0)}) - \frac{n}{m}\tilde{D}(y^{(0)}) \leq \max\left\{\frac{p}{q}, \frac{n}{m}\right\}\left(\tilde{P}(x^{(0)}) - \tilde{D}(y^{(0)})\right) \leq \max\left\{\frac{p}{q}, \frac{n}{m}\right\}\left(P(x^{(0)}) - D(y^{(0)})\right).$$

Using this inequality and the definition (A.3.17) of $\Delta^{(0)}$, we obtain

$$\Delta^{(0)} \leq \left(\frac{p}{2q\tau} + \frac{p\lambda}{2q}\right)\|x^{(0)} - x^\star\|^2 + \left(\frac{n}{2m\sigma} + \frac{\gamma}{2m}\right)\|y^{(0)} - y^\star\|^2 + \max\left\{\frac{p}{q}, \frac{n}{m}\right\}\left(P(x^{(0)}) - D(y^{(0)})\right)$$

Applying this inequality to the right hand size of (A.3.29), we obtain the conclusion of Theorem 5.4.2 by the new definition (5.4.9) of θ. ■

Appendix B

Convergence Analysis for DSCOVR

B.1 Proof of Theorem 6.3.1

We first prove two lemmas concerning the primal and dual proximal updates in Algorithm 7. Throughout this appendix, $\mathbf{E}_t[\cdot]$ denotes the conditional expectation taken with respect to the random indices j and l generated during the tth inner iteration in Algorithm 7, conditioned on all quantities available at the beginning of the tth iteration, including $w^{(t)}$ and $\alpha^{(t)}$. Whenever necessary, we also use the notation $j^{(t)}$ and $l^{(t)}$ to denote the random indices generated in the tth iteration.

Lemma 3 *For each $i = 1, \ldots, m$, let $u_i^{(t+1)} \in \mathbf{R}^{N_i}$ be a random variable and define*

$$\tilde{\alpha}_i^{(t+1)} = \mathbf{prox}_{\sigma_i f_i^*}\big(\alpha_i^{(t)} + \sigma_i u_i^{(t+1)}\big). \tag{B.1.1}$$

We choose an index j randomly from $\{1, \ldots, m\}$ with probability distribution $\{p_j\}_{j=1}^m$ and let

$$\alpha_i^{(t+1)} = \begin{cases} \tilde{\alpha}_i^{(t+1)} & \text{if } i = j, \\ \alpha_i^{(t)} & \text{otherwise.} \end{cases}$$

If each $u_i^{(t+1)}$ is independent of j and satisfies $\mathbf{E}_t\big[u_i^{(t+1)}\big] = X_{i:}w^{(t)}$ for $i = 1, \ldots, m$, then we have

$$\begin{aligned} & \sum_{i=1}^m \left(\frac{1}{p_i}\left(\frac{1}{2\sigma_i} + \gamma_i\right) - \gamma_i\right) \|\alpha_i^{(t)} - \alpha_i^\star\|^2 \\ \geq\ & \sum_{i=1}^m \frac{1}{p_i}\left(\frac{1}{2\sigma_i} + \gamma_i\right) \mathbf{E}_t[\|\alpha_i^{(t+1)} - \alpha_i^\star\|^2] + \sum_{i=1}^m \frac{1}{p_i}\left(\frac{1}{2\sigma_i} - \frac{1}{a_i}\right) \mathbf{E}_t[\|\alpha_i^{(t+1)} - \alpha_i^{(t)}\|^2] \\ & - \sum_{i=1}^m \frac{a_i}{4} \mathbf{E}_t\big[\|u_i^{(t+1)} - X_{i:}w^{(t)}\|^2\big] + \big\langle w^{(t)} - w^\star,\, X^T(\alpha^\star - \alpha^{(t)})\big\rangle \\ & - \sum_{i=1}^m \frac{1}{p_i} \mathbf{E}_t\Big[\Big\langle \alpha_i^{(t+1)} - \alpha_i^{(t)},\, X_{i:}(w^{(t)} - w^\star)\Big\rangle\Big], \end{aligned} \tag{B.1.2}$$

where $(w^\star, \alpha^\star)$ is the saddle point of $L(w, \alpha)$ defined in (6.2.1), and the a_i's are arbitrary positive numbers.

Proof: First, consider a fixed index $i \in \{1, \ldots, m\}$. The definition of $\tilde{\alpha}_i^{(t+1)}$ in (B.1.1) is equivalent to

$$\tilde{\alpha}_i^{(t+1)} = \operatorname*{argmin}_{\beta \in \mathbf{R}^{N_i}} \left\{ f_i^*(\beta) - \left\langle \beta, u_i^{(t+1)} \right\rangle + \frac{\|\beta - \alpha_i^{(t)}\|^2}{2\sigma_i} \right\}. \tag{B.1.3}$$

By assumption, $f_i^*(\beta)$ and $\frac{1}{2\sigma_i}\|\beta - \alpha_i^{(t)}\|^2$ are strongly convex with convexity parameters γ_i and $\frac{1}{\sigma_i}$ respectively. Therefore, the objective function in (B.1.3) is $(\frac{1}{\sigma_i} + \gamma_i)$-strongly convex, which implies

$$\begin{aligned} & \frac{\|\alpha_i^\star - \alpha_i^{(t)}\|^2}{2\sigma_i} - \left\langle \alpha_i^\star,\, u_i^{(t+1)} \right\rangle + f_i^*(\alpha_i^\star) \\ \geq\ & \frac{\|\tilde{\alpha}_i^{(t+1)} - \alpha_i^{(t)}\|^2}{2\sigma_i} - \left\langle \tilde{\alpha}_i^{(t+1)},\, u_i^{(t+1)} \right\rangle + f_i^*(\tilde{\alpha}_i^{(t+1)}) + \left(\frac{1}{\sigma_i} + \gamma_i\right) \frac{\|\tilde{\alpha}_i^{(t+1)} - \alpha_i^\star\|^2}{2} \end{aligned} \tag{B.1.4}$$

In addition, since $(w^\star, \alpha^\star)$ is the saddle-point of $L(w, \alpha)$, the function $f_i^*(\alpha_i) - \langle \alpha_i, X_{i:} w^\star \rangle$ is γ_i-strongly convex in α_i and attains its minimum at $\alpha_i^\star$. Thus we have

$$f_i^*\big(\tilde{\alpha}_i^{(t+1)}\big) - \left\langle \tilde{\alpha}_i^{(t+1)},\, X_{i:} w^\star \right\rangle \geq f_i^*(\alpha_i^\star) - \left\langle \alpha_i^\star,\, X_{i:} w^\star \right\rangle + \frac{\gamma_i}{2} \|\tilde{\alpha}_i^{(t+1)} - \alpha_i^\star\|^2.$$

Summing up the above two inequalities gives

$$\begin{aligned} \frac{\|\alpha_i^\star - \alpha_i^{(t)}\|^2}{2\sigma_i} \ &\geq\ \frac{\|\tilde{\alpha}_i^{(t+1)} - \alpha_i^{(t)}\|^2}{2\sigma_i} + \Big(\frac{1}{2\sigma_i} + \gamma_i\Big)\|\tilde{\alpha}_i^{(t+1)} - \alpha_i^\star\|^2 + \left\langle \alpha_i^\star - \tilde{\alpha}_i^{(t+1)},\, u_i^{(t+1)} - X_{i:} w^\star \right\rangle \\ &=\ \frac{\|\tilde{\alpha}_i^{(t+1)} - \alpha_i^{(t)}\|^2}{2\sigma_i} + \Big(\frac{1}{2\sigma_i} + \gamma_i\Big)\|\tilde{\alpha}_i^{(t+1)} - \alpha_i^\star\|^2 + \left\langle \alpha_i^\star - \tilde{\alpha}_i^{(t+1)},\, X_{i:}(w^{(t)} - w^\star) \right\rangle \\ &\quad + \left\langle \alpha_i^\star - \alpha_i^{(t)},\, u_i^{(t+1)} - X_{i:} w^{(t)} \right\rangle + \left\langle \alpha_i^{(t)} - \tilde{\alpha}_i^{(t+1)},\, u_i^{(t+1)} - X_{i:} w^{(t)} \right\rangle \\ &\geq\ \frac{\|\tilde{\alpha}_i^{(t+1)} - \alpha_i^{(t)}\|^2}{2\sigma_i} + \Big(\frac{1}{2\sigma_i} + \gamma_i\Big)\|\tilde{\alpha}_i^{(t+1)} - \alpha_i^\star\|^2 + \left\langle \alpha_i^\star - \tilde{\alpha}_i^{(t+1)},\, X_{i:}(w^{(t)} - w^\star) \right\rangle \\ &\quad + \left\langle \alpha_i^\star - \alpha_i^{(t)},\, u_i^{(t+1)} - X_{i:} w^{(t)} \right\rangle - \frac{\|\alpha_i^{(t)} - \tilde{\alpha}_i^{(t+1)}\|^2}{a_i} - \frac{a_i \|u_i^{(t+1)} - X_{i:} w^{(t)}\|^2}{4}, \end{aligned}$$

where in the last step we used Young's inequality with a_i being an arbitrary positive number. Taking conditional expectation $\mathbf{E}_t$ on both sides of the above inequality, and using the assumption $\mathbf{E}_t[u_i^{(t+1)}] = X_{i:} w^{(t)}$, we have

$$\begin{aligned} \frac{\|\alpha_i^\star - \alpha_i^{(t)}\|^2}{2\sigma_i} \geq\ & \frac{\mathbf{E}_t\big[\|\tilde{\alpha}_i^{(t+1)} - \alpha_i^{(t)}\|^2\big]}{2\sigma_i} + \Big(\frac{1}{2\sigma_i} + \gamma_i\Big)\mathbf{E}_t[\|\tilde{\alpha}_i^{(t+1)} - \alpha_i^\star\|^2] + \mathbf{E}_t\Big[\Big\langle \alpha_i^\star - \tilde{\alpha}_i^{(t+1)}, X_{i:}(w^{(t)} - w^\star)\Big\rangle\Big] \\ & - \frac{\mathbf{E}_t[\|\alpha_i^{(t)} - \tilde{\alpha}_i^{(t+1)}\|^2]}{a_i} - \frac{a_i \mathbf{E}_t[\|u_i^{(t+1)} - X_{i:} w^{(t)}\|^2]}{4}. \end{aligned} \tag{B.1.5}$$

Notice that each $\tilde{\alpha}_i^{(t+1)}$ depends on the random variable $u_i^{(t+1)}$ and is independent of the random index j. But $\alpha_i^{(t+1)}$ depends on both $u_i^{(t+1)}$ and j. Using the law of total expectation,

$$\mathbf{E}_t[\,\cdot\,] \;=\; \mathbf{P}(j=i)\mathbf{E}_t[\,\cdot\,|j=i] + \mathbf{P}(j\neq i)\mathbf{E}_t[\,\cdot\,|j\neq i],$$

we obtain

$$\begin{aligned}
\mathbf{E}_t[\alpha_i^{(t+1)}] &= p_i\mathbf{E}_t[\tilde{\alpha}_i^{(t+1)}] + (1-p_i)\alpha_i^{(t)}, && \text{(B.1.6)}\\
\mathbf{E}_t[\|\alpha_i^{(t+1)} - \alpha_i^{(t)}\|^2] &= p_i\mathbf{E}_t[\|\tilde{\alpha}_i^{(t+1)} - \alpha_i^{(t)}\|^2], && \text{(B.1.7)}\\
\mathbf{E}_t[\|\alpha_i^{(t+1)} - \alpha_i^{\star}\|^2] &= p_i\mathbf{E}_t[\|\tilde{\alpha}_i^{(t+1)} - \alpha_i^{\star}\|^2] + (1-p_i)\mathbf{E}_t[\|\alpha_i^{(t)} - \alpha_i^{\star}\|^2]. && \text{(B.1.8)}
\end{aligned}$$

Next, using the equalities (B.1.6), (B.1.7) and (B.1.8), we can replace each term in (B.1.5) containing $\tilde{\alpha}_i^{(t+1)}$ with terms that contain only $\alpha_i^{(t)}$ and $\alpha_i^{(t+1)}$. By doing so and rearranging terms afterwards, we obtain

$$\begin{aligned}
&\left(\frac{1}{p_i}\left(\frac{1}{2\sigma_i}+\gamma_i\right)-\gamma_i\right)\|\alpha_i^{(t)}-\alpha_i^{\star}\|^2\\
\geq\;& \frac{1}{p_i}\left(\frac{1}{2\sigma_i}+\gamma_i\right)\mathbf{E}_t[\|\alpha_i^{(t+1)}-\alpha_i^{\star}\|^2 + \frac{1}{p_i}\left(\frac{1}{2\sigma_i}-\frac{1}{a_i}\right)\mathbf{E}_t[\|\alpha_i^{(t+1)}-\alpha_i^{(t)}\|^2\\
&-\frac{a_i\mathbf{E}_t[\|u_i^{(t+1)}-X_{i:}w^{(t)}\|^2]}{4} + \left\langle \alpha_i^{\star}-\alpha_i^{(t)},\, X_{i:}(w^{(t)}-w^{\star})\right\rangle\\
&-\mathbf{E}_t\Big[\Big\langle\frac{1}{p_i}(\alpha_i^{(t+1)}-\alpha_i^{(t)}),\, X_{i:}(w^{(t)}-w^{\star})\Big\rangle\Big].
\end{aligned}$$

Summing up the above inequality for $i=1,\ldots,m$ gives the desired result in (B.1.2). ■

Lemma 4 *For each $k=1,\ldots,n$, let $v_k^{(t+1)}\in\mathbf{R}^{d_i}$ be a random variable and define*

$$\tilde{w}_k^{(t+1)} = \mathbf{prox}_{\tau_k g_k}\big(w_k^{(t)} - \tau_k v_k^{(t+1)}\big).$$

We choose an index l randomly from $\{1,\ldots,n\}$ with probability distribution $\{q_l\}_{l=1}^n$ and let

$$w_k^{(t+1)} = \begin{cases} \tilde{w}_k^{(t+1)} & \text{if } k=l,\\ w_k^{(t)} & \text{otherwise.}\end{cases}$$

If each $v_k^{(t+1)}$ is independent of l and satisfies $\mathbf{E}_t\big[v_k^{(t+1)}\big] = \frac{1}{m}(X_{:k})^T\alpha^{(t)}$, then we have

$$\begin{aligned}
&\sum_{k=1}^n\left(\frac{1}{q_k}\left(\frac{1}{2\tau_k}+\lambda\right)-\lambda\right)\|w_k^{(t)}-w_k^{\star}\|^2\\
\geq\;& \sum_{k=1}^n\frac{1}{q_k}\left(\frac{1}{2\tau_k}+\lambda\right)\mathbf{E}_t[\|w_k^{(t+1)}-w_k^{\star}\|^2] + \sum_{k=1}^n\frac{1}{q_k}\left(\frac{1}{2\tau_k}-\frac{1}{b_k}\right)\mathbf{E}_t[\|w_k^{(t+1)}-w_k^{(t)}\|^2]\\
&-\sum_{k=1}^n\frac{b_k}{4}\mathbf{E}_t\left[\left\|v_k^{(t+1)}-\frac{1}{m}(X_{:k})^T\alpha^{(t)}\right\|^2\right] + \frac{1}{m}\left\langle X(w^{(t)}-w^{\star}),\,\alpha^{(t)}-\alpha^{\star}\right\rangle\\
&+\sum_{k=1}^n\frac{1}{q_k}\mathbf{E}_t\left[\left\langle w_k^{(t+1)}-w_k^{(t)},\,\frac{1}{m}(X_{:k})^T(\alpha^{(t)}-\alpha^{\star})\right\rangle\right], \qquad \text{(B.1.9)}
\end{aligned}$$

where $(w^\star, \alpha^\star)$ is the saddle point of $L(w, \alpha)$ defined in (6.2.1), and the b_i's are arbitrary positive numbers.

Lemma 4 is similar to Lemma 3 and can be proved using the same techniques. Based on these two lemmas, we can prove the following proposition.

Proposition 3 *The t-th iteration within the s-th stage of Algorithm 7 guarantees*

$$\begin{aligned}
&\sum_{i=1}^{m} \frac{1}{m}\left[\frac{1}{p_i}\left(\frac{1}{2\sigma_i}+\gamma_i\right)-\gamma_i+\sum_{k=1}^{n}\frac{3\tau_k\|X_{ik}\|^2}{mp_i}\right]\|\alpha_i^{(t)}-\alpha_i^\star\|^2+\sum_{i=1}^{m}\sum_{k=1}^{n}\frac{2\tau_k\|X_{ik}\|^2}{m^2p_i}\|\bar{\alpha}_i^{(s)}-\alpha_i^\star\|^2\\
&+\sum_{k=1}^{n}\left[\frac{1}{q_k}\left(\frac{1}{2\tau_k}+\lambda\right)-\lambda+\sum_{i=1}^{m}\frac{3\sigma_i\|X_{ik}\|^2}{mq_k}\right]\|w_k^{(t)}-w_k^\star\|^2+\sum_{i=1}^{m}\sum_{k=1}^{n}\frac{2\sigma_i\|X_{ik}\|^2}{mq_k}\|\bar{w}_k^{(s)}-w_k^\star\|^2\\
\geq\ &\sum_{i=1}^{m}\frac{1}{mp_i}\left(\frac{1}{2\sigma_i}+\gamma_i\right)\mathbf{E}_t\big[\|\alpha_i^{(t+1)}-\alpha_i^\star\|^2\big]+\sum_{k=1}^{n}\frac{1}{q_k}\left(\frac{1}{2\tau_k}+\lambda\right)\mathbf{E}_t\big[\|w_k^{(t+1)}-w_k^\star\|^2\big]. \qquad \text{(B.1.10)}
\end{aligned}$$

Proof: Multiplying both sides of the inequality (B.1.2) by $\frac{1}{m}$ and adding to the inequality (B.1.9) gives

$$\begin{aligned}
&\sum_{i=1}^{m}\frac{1}{m}\left(\frac{1}{p_i}\left(\frac{1}{2\sigma_i}+\gamma_i\right)-\gamma_i\right)\|\alpha_i^{(t)}-\alpha_i^\star\|^2+\sum_{k=1}^{n}\left(\frac{1}{q_k}\left(\frac{1}{2\tau_k}+\lambda\right)-\lambda\right)\|w_k^{(t)}-w_k^\star\|^2\\
\geq\ &\sum_{i=1}^{m}\frac{1}{mp_i}\left(\frac{1}{2\sigma_i}+\gamma_i\right)\mathbf{E}_t[\|\alpha_i^{(t+1)}-\alpha_i^\star\|^2]+\sum_{k=1}^{n}\frac{1}{q_k}\left(\frac{1}{2\tau_k}+\lambda\right)\mathbf{E}_t[\|w_k^{(t+1)}-w_k^\star\|^2]\\
&+\sum_{i=1}^{m}\frac{1}{mp_i}\left(\frac{1}{2\sigma_i}-\frac{1}{a_i}\right)\mathbf{E}_t[\|\alpha_i^{(t+1)}-\alpha_i^{(t)}\|^2]+\sum_{k=1}^{n}\frac{1}{q_k}\left(\frac{1}{2\tau_k}-\frac{1}{b_k}\right)\mathbf{E}_t[\|w_k^{(t+1)}-w_k^{(t)}\|^2]\\
&-\sum_{k=1}^{n}\frac{b_k}{4}\mathbf{E}_t\left[\left\|v_k^{(t+1)}-\frac{1}{m}(X_{:k})^T\alpha^{(t)}\right\|^2\right]+\sum_{k=1}^{n}\frac{1}{q_k}\mathbf{E}_t\left[\left\langle w_k^{(t+1)}-w_k^{(t)},\ \frac{1}{m}(X_{:k})^T(\alpha^{(t)}-\alpha^\star)\right\rangle\right]\\
&-\sum_{i=1}^{m}\frac{a_i}{4m}\mathbf{E}_t[\|u_i^{(t+1)}-X_{i:}w^{(t)}\|^2]-\sum_{i=1}^{m}\frac{1}{mp_i}\mathbf{E}_t\Big[\left\langle\alpha_i^{(t+1)}-\alpha_i^{(t)},\ X_{i:}(w^{(t)}-w^\star)\right\rangle\Big]. \qquad \text{(B.1.11)}
\end{aligned}$$

We notice that the terms containing $\frac{1}{m}\left\langle X(w^{(t)}-w^\star),\ \alpha^{(t)}-\alpha^\star\right\rangle$ from (B.1.2) and (B.1.9) canceled each other. Next we bound the last four terms on the right-hand side of (B.1.11).

As in Algorithm 7, for each $i = 1, \ldots, m$, we define a random variable

$$u_i^{(t+1)} = \bar{u}_i^{(s)} - \frac{1}{q_l}X_{il}\bar{w}_l^{(s)} + \frac{1}{q_l}X_{il}w_l^{(t)},$$

which depends on the random index $l \in \{1, \ldots, n\}$. Taking expectation with respect to l yields

$$\mathbf{E}_t[u_i^{(t+1)}] = \sum_{k=1}^{n} q_k\left(\bar{u}_i^{(s)} - \frac{1}{q_k}X_{ik}\bar{w}_k^{(s)} + \frac{1}{q_k}X_{ik}w_k^{(t)}\right) = X_{i:}w^{(t)}, \qquad i = 1, 2, \ldots, m.$$

Therefore $u_i^{(t+1)}$ satisfies the assumption in Lemma 3. In order to bound its variance, we notice that

$$\sum_{k=1}^{n} q_k\left(\frac{1}{q_k}X_{ik}\bar{w}_k^{(s)} - \frac{1}{q_k}X_{ik}w_k^{(t)}\right) = X_{i:}\bar{w}^{(s)} - X_{i:}w^{(t)} = \bar{u}_i^{(s)} - X_{i:}w^{(t)}.$$

Using the relation between variance and the second moment, we have

$$\begin{aligned}
\mathbf{E}_t\big[\|u_i^{(t+1)} - X_{i:}w^{(t)}\|^2\big] &= \sum_{k=1}^{n} q_k \left\| \bar{u}_i^{(s)} - \frac{1}{q_k} X_{ik}\bar{w}_k^{(s)} + \frac{1}{q_k} X_{ik} w_k^{(t)} - X_{i:}w^{(t)} \right\|^2 \\
&= \sum_{k=1}^{n} \frac{1}{q_k} \left\| X_{ik}\bar{w}_k^{(s)} - X_{ik} w_k^{(t)} \right\|^2 - \|\bar{u}_i^{(s)} - X_{i:}w^{(t)}\|^2 \\
&\leq \sum_{k=1}^{n} \frac{1}{q_k} \left\| X_{ik}(\bar{w}_k^{(s)} - w_k^{(t)}) \right\|^2 \\
&\leq \sum_{k=1}^{n} \frac{2\|X_{ik}\|^2}{q_k} \left(\|\bar{w}_k^{(s)} - w_k^\star\|^2 + \|w_k^{(t)} - w_k^\star\|^2 \right). \qquad \text{(B.1.12)}
\end{aligned}$$

Similarly, for $k = 1, \ldots, n$, we have

$$\mathbf{E}_t[v_k^{(t+1)}] = \sum_{i=1}^{m} p_i \left(\bar{v}_k^{(s)} - \frac{1}{p_i}\frac{1}{m}(X_{ik})^T \bar{\alpha}_i^{(s)} + \frac{1}{p_i}\frac{1}{m}(X_{ik})^T \alpha_i^{(t)} \right) = \frac{1}{m}(X_{:k})^T \alpha^{(t)}.$$

Therefore $v_k^{(t+1)}$ satisfies the assumption in Lemma 4. Furthermore, we have

$$\begin{aligned}
\mathbf{E}_t\left[\left\| v_k^{(t+1)} - \frac{1}{m}(X_{:k})^T \alpha^{(t)} \right\|^2 \right] &= \sum_{i=1}^{m} p_i \left\| \bar{v}_k^{(s)} - \frac{1}{p_i}\frac{1}{m}(X_{ik})^T \bar{\alpha}_i^{(s)} + \frac{1}{p_i}\frac{1}{m}(X_{ik})^T \alpha_i^{(t)} - \frac{1}{m}(X_{:k})^T \alpha^{(t)} \right\|^2 \\
&= \sum_{i=1}^{m} \frac{1}{p_i} \left\| \frac{1}{m}(X_{ik})^T \bar{\alpha}_i^{(s)} - \frac{1}{m}(X_{ik})^T \alpha_i^{(t)} \right\|^2 - \left\| \bar{v}_k^{(s)} - \frac{1}{m}(X_{:k})^T \alpha^{(t)} \right\|^2 \\
&\leq \sum_{i=1}^{m} \frac{1}{p_i} \left\| \frac{1}{m}(X_{ik})^T \left(\bar{\alpha}_i^{(s)} - \alpha_i^{(t)} \right) \right\|^2 \\
&\leq \sum_{i=1}^{m} \frac{2\|X_{ik}\|^2}{p_i m^2} \left(\|\bar{\alpha}_i^{(s)} - \alpha_i^\star\|^2 + \|\alpha_i^{(t)} - \alpha_i^\star\|^2 \right). \qquad \text{(B.1.13)}
\end{aligned}$$

Now we consider the two terms containing inner products in (B.1.11). Using the conditional expectation relation (B.1.6), we have

$$\begin{aligned}
\mathbf{E}_t\left[-\left\langle \alpha_i^{(t+1)} - \alpha_i^{(t)},\, X_{i:}(w^{(t)} - w^\star) \right\rangle \right] &= p_i \mathbf{E}_t\left[-\left\langle \tilde{\alpha}_i^{(t+1)} - \alpha_i^{(t)},\, X_{i:}(w^{(t)} - w^\star) \right\rangle \right] \\
&\geq p_i \mathbf{E}_t\left[-\frac{1}{c_i}\|\tilde{\alpha}_i^{(t+1)} - \alpha_i^{(t)}\|^2 - \frac{c_i}{4}\|X_{i:}(w^{(t)} - w^\star)\|^2 \right] \\
&= -\frac{p_i}{c_i}\mathbf{E}_t\big[\|\tilde{\alpha}_i^{(t+1)} - \alpha_i^{(t)}\|^2\big] - \frac{c_i p_i}{4}\|X_{i:}(w^{(t)} - w^\star)\|^2 \\
&= -\frac{1}{c_i}\mathbf{E}_t\big[\|\alpha_i^{(t+1)} - \alpha_i^{(t)}\|^2\big] - \frac{c_i p_i}{4}\|X_{i:}(w^{(t)} - w^\star)\|^2, \qquad \text{(B.1.14)}
\end{aligned}$$

where we used Young's inequality with c_i being an arbitrary positive number, and the last equality used (B.1.7). We note that for any n vectors $z_1, \ldots, z_n \in \mathbf{R}^{N_i}$, it holds that

$$\Big\|\sum_{k=1}^n z_k\Big\|^2 \le \sum_{k=1}^n \frac{1}{q_k}\|z_k\|^2.$$

To see this, we let $z_{k,j}$ denote the jth component of z_k and use the Cauchy-Schwarz inequality:

$$\begin{aligned}
\Big\|\sum_{k=1}^n z_k\Big\|^2 &= \sum_{j=1}^{N_i}\left(\sum_{k=1}^n z_{k,j}\right)^2 = \sum_{j=1}^{N_i}\left(\sum_{k=1}^n \frac{z_{k,j}}{\sqrt{q_k}}\sqrt{q_k}\right)^2 \\
&\le \sum_{j=1}^{N_i}\left(\sum_{k=1}^n\Big(\frac{z_{k,j}}{\sqrt{q_k}}\Big)^2\right)\left(\sum_{k=1}^n\big(\sqrt{q_k}\big)^2\right) \\
&= \sum_{j=1}^{N_i}\left(\sum_{k=1}^n \frac{z_{k,j}^2}{q_k}\right) = \sum_{k=1}^n \frac{1}{q_k}\sum_{j=1}^{N_i} z_{k,j}^2 = \sum_{k=1}^n \frac{1}{q_k}\|z_k\|^2.
\end{aligned}$$

Applying this inequality to the vector $X_{i:}(w^{(t)} - w^\star) = \sum_{k=1}^n X_{ik}(w_k^{(t)} - w_k^\star)$, we get

$$\|X_{i:}(w^{(t)} - w^\star)\|^2 \le \sum_{k=1}^n \frac{1}{q_k}\|X_{ik}(w_k^{(t)} - w_k^\star)\|^2.$$

Therefore we can continue the inequality (B.1.14), for each $i = 1, \ldots, m$, as

$$\begin{aligned}
&\mathbf{E}_t\Big[-\Big\langle \alpha_i^{(t+1)} - \alpha_i^{(t)},\, X_{i:}(w^{(t)} - w^\star)\Big\rangle\Big] \\
\ge\ & -\frac{1}{c_i}\mathbf{E}_t\big[\|\alpha_i^{(t+1)} - \alpha_i^{(t)}\|^2\big] - \frac{c_i p_i}{4}\sum_{k=1}^n \frac{1}{q_k}\|X_{ik}(w_k^{(t)} - w_k^\star)\|^2 \\
\ge\ & -\frac{1}{c_i}\mathbf{E}_t\big[\|\alpha_i^{(t+1)} - \alpha_i^{(t)}\|^2\big] - \frac{c_i p_i}{4}\sum_{k=1}^n \frac{1}{q_k}\|X_{ik}\|^2\|w_k^{(t)} - w_k^\star\|^2. \qquad \text{(B.1.15)}
\end{aligned}$$

Using similarly arguments, we can obtain, for each $k = 1, \ldots, n$ and arbitrary $h_k > 0$,

$$\begin{aligned}
&\mathbf{E}_t\Big[\Big\langle w_k^{(t+1)} - w_k^{(t)},\, \frac{1}{m}(X_{:k})^T(\alpha^{(t)} - \alpha^\star)\Big\rangle\Big] \\
\ge\ & -\frac{1}{h_k}\mathbf{E}_t\big[\|w_k^{(t+1)} - w_k^{(t)}\|^2\big] - \frac{h_k q_k}{4m^2}\sum_{i=1}^m \frac{1}{p_i}\|X_{ik}\|^2\|\alpha_i^{(t)} - \alpha_i^\star\|^2. \qquad \text{(B.1.16)}
\end{aligned}$$

Applying the bounds in (B.1.12), (B.1.13), (B.1.15) and (B.1.16) to (B.1.11) and rearranging terms, we have

$$
\begin{aligned}
&\sum_{i=1}^{m} \frac{1}{m}\left[\frac{1}{p_i}\left(\frac{1}{2\sigma_i}+\gamma_i\right)-\gamma_i+\sum_{k=1}^{n}\frac{b_k\|X_{ik}\|^2}{2mp_i}+\sum_{k=1}^{n}\frac{h_k\|X_{ik}\|^2}{4mp_i}\right]\|\alpha_i^{(t)}-\alpha_i^\star\|^2\\
&+\sum_{k=1}^{n}\left[\frac{1}{q_k}\left(\frac{1}{2\tau_k}+\lambda\right)-\lambda+\sum_{i=1}^{m}\frac{a_i\|X_{ik}\|^2}{2mq_k}+\sum_{i=1}^{m}\frac{c_i\|X_{ik}\|^2}{4mq_k}\right]\|w_k^{(t)}-w_k^\star\|^2\\
&+\sum_{i=1}^{m}\sum_{k=1}^{n}\frac{b_k\|X_{ik}\|^2}{2m^2p_i}\|\bar{\alpha}_i^{(s)}-\alpha_i^\star\|^2+\sum_{i=1}^{m}\sum_{k=1}^{n}\frac{a_i\|X_{ik}\|^2}{2mq_k}\|\bar{w}_k^{(s)}-w_k^\star\|^2\\
\geq\ &\sum_{i=1}^{m}\frac{1}{mp_i}\left(\frac{1}{2\sigma_i}+\gamma_i\right)\mathbf{E}_t[\|\alpha_i^{(t+1)}-\alpha_i^\star\|^2]+\sum_{k=1}^{n}\frac{1}{q_k}\left(\frac{1}{2\tau_k}+\lambda\right)\mathbf{E}_t[\|w_k^{(t+1)}-w_k^\star\|^2]\\
&+\sum_{i=1}^{m}\frac{1}{mp_i}\left(\frac{1}{2\sigma_i}-\frac{1}{a_i}-\frac{1}{c_i}\right)\mathbf{E}_t[\|\alpha_i^{(t+1)}-\alpha_i^{(t)}\|^2]\\
&+\sum_{k=1}^{n}\frac{1}{q_k}\left(\frac{1}{2\tau_k}-\frac{1}{b_k}-\frac{1}{h_k}\right)\mathbf{E}_t[\|w_k^{(t+1)}-w_k^{(t)}\|^2].
\end{aligned}
$$

The desired result (B.1.10) is obtained by choosing $a_i = c_i = 4\sigma_i$ and $b_k = h_k = 4\tau_k$. ■

Finally, we are ready to prove Theorem 6.3.1. Let $\theta \in (0,1)$ be a parameter to be determined later, and let Γ and η be two constants such that

$$
\Gamma \ \geq\ \max_{i,k}\left\{\frac{1}{p_i}\left(1+\frac{3\|X_{ik}\|^2}{2\theta q_k\lambda\gamma_i}\right),\ \frac{1}{q_k}\left(1+\frac{3n\|X_{ik}\|^2}{2\theta mp_i\lambda\gamma_i}\right)\right\}, \tag{B.1.17}
$$

$$
\eta \ =\ 1-\frac{1-\theta}{\Gamma}. \tag{B.1.18}
$$

It is easy to check that $\Gamma > 1$ and $\eta \in (0,1)$. By the choices of σ_i and τ_k in (6.3.5) and (6.3.6) respectively, we have

$$
\frac{1}{p_i}\left(1+\frac{1}{2\sigma_i\gamma_i}\right)=\frac{1}{q_k}\left(1+\frac{1}{2\tau_k\lambda}\right)=\Gamma, \tag{B.1.19}
$$

for all $i = 1,\ldots,m$ and $k = 1,\ldots,n$. Comparing the above equality with the definition of Γ in (B.1.17), we have

$$
\frac{3\|X_{ik}\|^2}{2\theta q_k\lambda\gamma_i}\leq\frac{1}{2\sigma_i\gamma_i}\qquad\text{and}\qquad\frac{3n\|X_{ik}\|^2}{2\theta mp_i\lambda\gamma_i}\leq\frac{1}{2\tau_k\lambda},
$$

which implies

$$
\frac{3\sigma_i\|X_{ik}\|^2}{q_k}\leq\theta\lambda\qquad\text{and}\qquad\frac{3n\tau_k\|X_{ik}\|^2}{mp_i}\leq\theta\gamma_i,
$$

for all $i = 1, \ldots, m$ and $k = 1, \ldots, n$. Therefore, we have

$$\sum_{k=1}^{n} \frac{3\tau_k \|X_{ik}\|^2}{mp_i} = \frac{1}{n}\sum_{k=1}^{n} \frac{3n\tau_k \|X_{ik}\|^2}{mp_i} \leq \frac{1}{n}\sum_{k=1}^{n} \theta\gamma_i = \theta\gamma_i, \qquad i = 1, \ldots, m, \tag{B.1.20}$$

$$\sum_{i=1}^{m} \frac{3\sigma_i \|X_{ik}\|^2}{mq_k} = \frac{1}{m}\sum_{i=1}^{m} \frac{3\sigma_i \|X_{ik}\|^2}{q_k} \leq \frac{1}{m}\sum_{i=1}^{m} \theta\lambda = \theta\lambda, \qquad k = 1, \ldots, n. \tag{B.1.21}$$

Now we consider the inequality (B.1.10), and examine the ratio between the coefficients of $\|\alpha_i^{(t)} - \alpha_i^\star\|^2$ and $\mathbf{E}_t[\|\alpha_i^{(t+1)} - \alpha_i^\star\|^2]$. Using (B.1.20) and (B.1.19), we have

$$\frac{\frac{1}{p_i}\left(\frac{1}{2\sigma_i} + \gamma_i\right) - \gamma_i + \sum_{k=1}^{n} \frac{3\tau_k \|X_{ik}\|^2}{mp_i}}{\frac{1}{p_i}\left(\frac{1}{2\sigma_i} + \gamma_i\right)} \leq 1 - \frac{(1-\theta)\gamma_i}{\frac{1}{p_i}\left(\frac{1}{2\sigma_i} + \gamma_i\right)} = 1 - \frac{1-\theta}{\Gamma} = \eta. \tag{B.1.22}$$

Similarly, the ratio between the coefficients of $\|w_k^{(t)} - w_k^\star\|^2$ and $\mathbf{E}_t[\|w_k^{(t+1)} - w_k^\star\|^2]$ can be bounded using (B.1.21) and (B.1.19):

$$\frac{\frac{1}{q_k}\left(\frac{1}{2\tau_k} + \lambda\right) - \lambda + \sum_{i=1}^{m} \frac{3\sigma_i \|X_{ik}\|^2}{mq_k}}{\frac{1}{q_k}\left(\frac{1}{2\tau_k} + \lambda\right)} \leq 1 - \frac{(1-\theta)\lambda}{\frac{1}{q_k}\left(\frac{1}{2\tau_k} + \lambda\right)} = 1 - \frac{1-\theta}{\Gamma} = \eta. \tag{B.1.23}$$

In addition, the ratio between the coefficients of $\|\bar{\alpha}_i^{(s)} - \alpha_i^\star\|^2$ and $\mathbf{E}_t[\|\alpha_i^{(t+1)} - \alpha_i^\star\|^2]$ and that of $\|\bar{w}_k^{(s)} - w_k^\star\|^2$ and $\mathbf{E}_t[\|w_k^{(t+1)} - w_k^\star\|^2]$ can be bounded as

$$\frac{\sum_{k=1}^{k} \frac{2\tau_k \|X_{ik}\|^2}{mp_i}}{\frac{1}{p_i}\left(\frac{1}{2\sigma_i} + \gamma_i\right)} \leq \frac{(2/3)\theta\gamma_i}{\frac{1}{p_i}\left(\frac{1}{2\sigma_i} + \gamma_i\right)} = \frac{(2/3)\theta}{\Gamma} = \frac{2\theta(1-\eta)}{3(1-\theta)}, \tag{B.1.24}$$

$$\frac{\sum_{i=1}^{m} \frac{3\sigma_i \|X_{ik}\|^2}{mq_k}}{\frac{1}{q_k}\left(\frac{1}{2\tau_k} + \lambda\right)} \leq \frac{(2/3)\theta\lambda}{\frac{1}{q_k}\left(\frac{1}{2\tau_k} + \lambda\right)} = \frac{(2/3)\theta}{\Gamma} = \frac{2\theta(1-\eta)}{3(1-\theta)}. \tag{B.1.25}$$

Using (B.1.19) and the four inequalities (B.1.22), (B.1.23), (B.1.24) and (B.1.25), we conclude that the inequality (B.1.10) implies

$$\begin{aligned}
&\eta \sum_{i=1}^{m} \frac{\Gamma\gamma_i}{m} \|\alpha_i^{(t)} - \alpha_i^\star\|^2 + \frac{2\theta(1-\eta)}{3(1-\theta)} \sum_{i=1}^{m} \frac{\Gamma\gamma_i}{m} \|\bar{\alpha}_i^{(s)} - \alpha_i^\star\|^2 \\
&+\eta \sum_{k=1}^{n} \Gamma\lambda \|w_k^{(t)} - w_k^\star\|^2 + \frac{2\theta(1-\eta)}{3(1-\theta)} \sum_{k=1}^{n} \Gamma\lambda \|\bar{w}_k^{(s)} - w_k^\star\|^2 \\
\geq\ & \sum_{i=1}^{m} \frac{\Gamma\gamma_i}{m} \mathbf{E}_t[\|\alpha_i^{(t+1)} - \alpha_i^\star\|^2] + \sum_{k=1}^{n} \Gamma\lambda \mathbf{E}_t[\|w_k^{(t+1)} - w_k^\star\|^2].
\end{aligned}$$

Using the definite of $\Omega(\cdot)$ in (6.3.3), the inequality above is equivalent to

$$\begin{aligned}
&\eta\, \Omega\big(w^{(t)} - w^\star, \alpha^{(t)} - \alpha^\star\big) + \frac{2\theta(1-\eta)}{3(1-\theta)} \Omega\big(\bar{w}^{(s)} - w^\star, |-|\, \alpha^\star\big) \\
\geq\ & \mathbf{E}_t\big[\Omega\big(w^{(t+1)} - w^\star, \alpha^{(t+1)} - \alpha^\star\big)\big]\,.
\end{aligned} \tag{B.1.26}$$

To simplify further derivation, we define

$$\begin{aligned}\Delta^{(t)} &= \mathbf{E}\left[\Omega\big(w^{(t)}-w^\star,\alpha^{(t)}-\alpha^\star\big)\right],\\ \bar{\Delta}^{(s)} &= \mathbf{E}\left[\Omega\big(\bar{w}^{(s)}-w^\star,|-|\,\alpha^\star\big)\right],\end{aligned}$$

where the expectation is taken with respect to all randomness in the sth stage, that is, the random variables $\{(j^{(0)},l^{(0)}),(j^{(1)},l^{(1)}),\ldots,(j^{(M-1)},l^{(M-1)})\}$. Then the inequality (B.1.26) implies

$$\frac{2\theta(1-\eta)}{3(1-\theta)}\bar{\Delta}^{(s)}+\eta\Delta^{(t)}\geq\Delta^{(t+1)}.$$

Dividing both sides of the above inequality by η^{t+1} gives

$$\frac{2\theta(1-\eta)}{3(1-\theta)}\frac{\bar{\Delta}^{(s)}}{\eta^{t+1}}+\frac{\Delta^{(t)}}{\eta^t}\geq\frac{\Delta^{(t+1)}}{\eta^{t+1}}.$$

Summing for $t=0,1,,\ldots,M-1$ gives

$$\left(\frac{1}{\eta}+\frac{1}{\eta^2}+\cdots+\frac{1}{\eta^M}\right)\frac{2\theta(1-\eta)}{3(1-\theta)}\bar{\Delta}^{(s)}+\Delta^{(0)}\geq\frac{\Delta^{(T)}}{\eta^M},$$

which further leads to

$$(1-\eta^M)\frac{2\theta}{3(1-\theta)}\bar{\Delta}^{(s)}+\eta^M\Delta^{(0)}\geq\Delta^{(M)}.$$

Now choosing $\theta=1/3$ and using the relation $\bar{\Delta}^{(s)}=\Delta^{(0)}$ for each stage, we obtain

$$\left(\frac{1}{3}+\frac{2}{3}\eta^M\right)\Delta^{(0)}\geq\Delta^{(M)}.$$

Therefore if we choose M large enough such that $\eta^M\leq\frac{1}{2}$, then

$$\Delta^{(M)}\leq\frac{2}{3}\Delta^{(0)},\quad\text{or equivalently,}\quad\bar{\Delta}^{(s+1)}\leq\frac{2}{3}\bar{\Delta}^{(s)}.$$

The condition $\eta^M\leq\frac{1}{2}$ is equivalent to $M\geq\frac{\log(2)}{\log(1/\eta)}$, which can be guaranteed by

$$M\geq\frac{\log(2)}{1-\eta}=\frac{\log(2)}{1-\theta}\Gamma=\frac{3\log(2)}{2}\Gamma=\log(\sqrt{8})\Gamma.$$

To further simplify, it suffices to have $M\geq\log(3)\Gamma$. Finally, we notice that $\bar{\Delta}^{(s+1)}\leq(2/3)\bar{\Delta}^{(s)}$ implies $\bar{\Delta}^{(s)}\leq(2/3)^s\,\bar{\Delta}^{(0)}$. which is the desired result in Theorem 6.3.1.

B.1.1 Alternative bounds and step sizes

Alternatively, we can let Γ to satisfy

$$\Gamma \geq \max_{i,k} \left\{ \frac{1}{p_i} \left(1 + \frac{3\|X_{:k}\|_F^2}{2\theta m q_k \lambda \gamma_i} \right), \frac{1}{q_k} \left(1 + \frac{3\|X_{i:}\|_F^2}{2\theta m p_i \lambda \gamma_i} \right) \right\}, \tag{B.1.27}$$

where $\|\cdot\|_F$ denotes the Frobenius norm of a matrix. Then by choosing σ_i and τ_k that satisfy (B.1.19), we have

$$\frac{3\sigma_i \|X_{:k}\|_F^2}{m q_k} \leq \theta\lambda \qquad \text{and} \qquad \frac{3\tau_k \|X_{i:}\|_F^2}{m p_i} \leq \theta\gamma_i,$$

We can bound the left-hand sides in (B.1.20) and (B.1.21) using Hölder's inequality, which results in

$$\sum_{k=1}^{n} \frac{3\tau_k \|X_{ik}\|^2}{m p_i} \leq \sum_{k=1}^{n} \frac{3\tau_k \|X_{ik}\|_F^2}{m p_i} \leq \frac{3\max_k\{\tau_k\}\|X_{i:}\|_F^2}{m p_i} \leq \theta\gamma_i, \qquad i = 1, \ldots, m, \tag{B.1.28}$$

$$\sum_{i=1}^{m} \frac{3\sigma_i \|X_{ik}\|^2}{m q_k} \leq \sum_{i=1}^{m} \frac{3\sigma_i \|X_{ik}\|_F^2}{m q_k} \leq \frac{3\max_i\{\sigma_i\}\|X_{:k}\|_F^2}{m q_k} \leq \theta\lambda, \qquad k = 1, \ldots, n. \tag{B.1.29}$$

The rest of the proof hold without any change. Setting $\theta = 1/3$ gives the condition on Γ in (6.3.8).

In Theorem 6.3.1 and the proof above, we choose Γ as a uniform bound over all combinations of (i, k) in order to obtain a uniform convergence rates on all blocks of the primal and dual variables $w_k^{(t)}$ and $\alpha_i^{(t)}$, so we have a simple conclusion as in (6.3.7). In practice, we can use different bounds on different blocks and choose step sizes to allow them to converge at different rates.

For example, we can choose the step sizes σ_i and τ_k such that

$$\begin{aligned}
\frac{1}{p_i}\left(1 + \frac{1}{2\sigma_i\gamma_i}\right) &= \max_k \left\{ \frac{1}{p_i} \left(1 + \frac{3\|X_{:k}\|_F^2}{2\theta m q_k \lambda \gamma_i} \right) \right\}, \qquad i = 1, \ldots, m, \\
\frac{1}{q_k}\left(1 + \frac{1}{2\tau_k\lambda}\right) &= \max_i \left\{ \frac{1}{q_k} \left(1 + \frac{3\|X_{i:}\|_F^2}{2\theta m p_i \lambda \gamma_i} \right) \right\}, \qquad k = 1, \ldots, n.
\end{aligned}$$

Then the inequalities (B.1.28) and (B.1.29) still hold, and we can still show linear convergence with a similar rate. In this case, the step sizes are chosen as

$$\begin{aligned}
\sigma_i &= \min_k \left\{ \frac{\theta m q_k \lambda}{3\|X_{:k}\|_F^2} \right\}, \qquad i = 1, \ldots, m, \\
\tau_k &= \min_i \left\{ \frac{\theta m p_i \gamma_i}{3\|X_{i:}\|_F^2} \right\}, \qquad k = 1, \ldots, n.
\end{aligned}$$

If we choose the probabilities to be proportional to the norms of the data blocks, i.e.,

$$p_i = \frac{\|X_{i:}\|_F^2}{\|X\|_F^2}, \qquad q_k = \frac{\|X_{:k}\|_F^2}{\|X\|_F^2},$$

then we have

$$\sigma_i = \frac{\theta m \lambda}{3\|X\|_F^2}, \qquad \tau_k = \frac{\theta m \gamma_i}{3\|X\|_F^2}.$$

If we further normalize the rows of X, and let R be the norm of each row, then (with $\theta = 1/3$)

$$\sigma_i = \frac{\theta\lambda}{3R^2}\frac{m}{N} = \frac{\lambda}{9R^2}\frac{m}{N}, \qquad \tau_k = \frac{\theta\gamma_i}{3R^2}\frac{m}{N} = \frac{\gamma_i}{9R^2}\frac{m}{N}.$$

For distributed ERM, we have $\gamma_i = \frac{N}{m}\gamma$, thus $\tau_k = \frac{\gamma}{9R^2}$ as in (6.3.11).

B.2 Proof of Theorem 6.3.2

Consider the following saddle-point problem with doubly separable structure:

$$\min_{w\in\mathbf{R}^D} \max_{\alpha\in\mathbf{R}^N} \left\{ L(w,\alpha) \equiv \frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{n} \alpha_i^T X_{ik} w_k - \frac{1}{m}\sum_{i=1}^{m} f_i^*(\alpha_i) + \sum_{k=1}^{n} g_k(w_k) \right\}. \tag{B.2.1}$$

Under Assumption 2, L has a unique saddle point $(w^\star, \alpha^\star)$. We define

$$\widetilde{P}_k(w_k) \equiv \frac{1}{m}(\alpha^\star)^T X_{:k} w_k + g_k(w_k) - \frac{1}{m}(\alpha^\star)^T X_{:k} w_k^\star - g_k(w_k^\star), \qquad k = 1,\ldots,n, \tag{B.2.2}$$

$$\widetilde{D}_i(\alpha_i) \equiv \frac{1}{m}\left(\alpha_i^T X_{i:} w^\star - f_i^*(\alpha_i) - (\alpha_i^\star)^T X_{i:} w^\star + f_i^*(\alpha_i^\star)\right), \qquad i = 1,\ldots,m. \tag{B.2.3}$$

We note that $w_k^\star$ is the minimizer of $\widetilde{P}_k$ with $\widetilde{P}_k(w_k^\star) = 0$ and $\alpha_i^\star$ is the maximizer of $\widetilde{D}_i$ with $\widetilde{D}_i(\alpha_i^\star) = 0$. Moreover, by the assumed strong convexity,

$$\widetilde{P}_k(w_k) \geq \frac{\lambda}{2}\|w_k - w_k^\star\|^2, \qquad \widetilde{D}_i(\alpha_i) \leq -\frac{\gamma_i}{2m}\|\alpha_i - \alpha_i^\star\|^2. \tag{B.2.4}$$

Moreover, we have the following lower bound for the duality gap $P(w) - D(\alpha)$:

$$\sum_{k=1}^{n}\widetilde{P}_k(w_k) - \sum_{i=1}^{m}\widetilde{D}_i(\alpha_i) = L(w,\alpha^\star) - L(w^\star,\alpha) \leq P(w) - D(\alpha). \tag{B.2.5}$$

We can also use them to derive an upper bound for the duality gap, as in the following lemma.

Lemma 5 *Suppose Assumption 2 holds. Let $(w^\star, \alpha^\star)$ be the saddle-point of $L(w,\alpha)$ and define*

$$P(w) = \sup_\alpha L(w,\alpha), \qquad D(\alpha) = \inf_w L(w,\alpha).$$

Then we have

$$P(w) - D(\alpha) \leq L(w,\alpha^\star) - L(w^\star,\alpha) + \left(\frac{1}{m}\sum_{i=1}^{m}\frac{\|X_{i:}\|^2}{2\gamma_i}\right)\|w - w^\star\|^2 + \frac{\|X\|^2}{2m^2\lambda}\|\alpha - \alpha^\star\|^2.$$

Proof: By definition, the primal function can be written as $P(w) = F(w) + g(w)$, where

$$F(w) = \frac{1}{m}\sum_{i=1}^{m} f_i(X_{i:}w) = \frac{1}{m}\max_{\alpha}\left\{\alpha^T X w - \sum_{i=1}^{m} f_i^*(\alpha_i)\right\}.$$

From the optimality conditions satisfied by the saddle point $(w^\star, \alpha^\star)$, we have

$$\nabla F(w^\star) = \frac{1}{m}X^T\alpha^\star.$$

By assumption, $\nabla F(w)$ is Lipschitz continuous with smooth constant $\frac{1}{m}\sum_{i=1}^{m}\frac{\|X_{i:}\|^2}{\gamma_i}$, which implies

$$\begin{aligned}
F(w) &\leq F(w^\star) + \langle \nabla F(w^\star),\, w - w^\star\rangle + \left(\frac{1}{m}\sum_{i=1}^{m}\frac{\|X_{i:}\|^2}{2\gamma_i}\right)\|w - w^\star\|^2 \\
&= \frac{1}{m}\left((\alpha^\star)^T X w^\star - \sum_{i=1}^{m} f_i^*(\alpha_i^\star)\right) + \frac{1}{m}(\alpha^\star)^T X(w - w^\star) + \left(\frac{1}{m}\sum_{i=1}^{m}\frac{\|X_{i:}\|^2}{2\gamma_i}\right)\|w - w^\star\|^2 \\
&= \frac{1}{m}\left((\alpha^\star)^T X w - \sum_{i=1}^{m} f_i^*(\alpha_i^\star)\right) + \left(\frac{1}{m}\sum_{i=1}^{m}\frac{\|X_{i:}\|^2}{2\gamma_i}\right)\|w - w^\star\|^2.
\end{aligned}$$

Therefore,

$$\begin{aligned}
P(w) &= F(w) + g(w) \\
&\leq \frac{1}{m}(\alpha^\star)^T X w - \frac{1}{m}\sum_{i=1}^{m} f_i^*(\alpha_i^\star) + g(w) + \left(\frac{1}{m}\sum_{i=1}^{m}\frac{\|X_{i:}\|^2}{2\gamma_i}\right)\|w - w^\star\|^2 \\
&= L(w, \alpha^\star) + \left(\frac{1}{m}\sum_{i=1}^{m}\frac{\|X_{i:}\|^2}{2\gamma_i}\right)\|w - w^\star\|^2.
\end{aligned}$$

Using similar arguments, especially that $\nabla g^*(\alpha)$ has Lipschitz constant $\frac{\|X\|}{m^2\lambda}$, we can show that

$$D(\alpha) \;\geq\; L(w^\star, \alpha) - \frac{\|X\|^2}{2m^2\lambda}\|\alpha - \alpha^\star\|^2.$$

Combining the last two inequalities gives the desired result. ■

The rest of the proof follow similar steps as in the proof of Theorem 6.3.1. The next two lemmas are variants of Lemmas 3 and 4.

Lemma 6 *Under the same assumptions and setup in Lemma 3, we have*

$$\begin{aligned}
&\sum_{i=1}^{m}\left(\frac{1}{p_i}\left(\frac{1}{2\sigma_i} + \frac{\gamma_i}{2}\right) - \frac{\gamma_i}{2}\right)\|\alpha_i^{(t)} - \alpha_i^\star\|^2 - \sum_{i=1}^{m}\left(\frac{1}{p_i} - 1\right) m\widetilde{D}_i(\alpha_i^{(t)}) \\
\geq\; &\sum_{i=1}^{m}\frac{1}{p_i}\left(\frac{1}{2\sigma_i} + \frac{\gamma_i}{2}\right)\mathbf{E}_t[\|\alpha_i^{(t+1)} - \alpha_i^\star\|^2] + \sum_{i=1}^{m}\frac{1}{2p_i\sigma_i}\mathbf{E}_t[\|\alpha_i^{(t+1)} - \alpha_i^{(t)}\|^2] - \sum_{i=1}^{m}\frac{m}{p_i}\mathbf{E}_t\big[\widetilde{D}_i(\alpha_i^{(t+1)})\big] \\
&+ \left\langle w^{(t)} - w^\star,\, X^T(\alpha^\star - \alpha^{(t)})\right\rangle - \sum_{i=1}^{m}\frac{1}{p_i}\mathbf{E}_t\left[\left\langle \alpha_i^{(t+1)} - \alpha_i^{(t)},\, u_i^{(t+1)} - X_{i:}w^\star\right\rangle\right]. \qquad \text{(B.2.6)}
\end{aligned}$$

Proof: We start by taking conditional expectation $\mathbf{E}_t$ on both sides of the inequality (B.1.4), and would like to replace every term containing $\tilde{\alpha}_i^{(t+1)}$ with terms that contain only $\alpha_i^{(t)}$ and $\alpha_i^{(t+1)}$. In addition to the relations in (B.1.6), (B.1.7) and (B.1.8), we also need

$$\mathbf{E}_t\big[f_i^*(\alpha_i^{(t+1)})\big] \;=\; p_i f_i^*(\tilde{\alpha}_i^{(t+1)}) + (1-p_i) f_i^*(\alpha_i^{(t)}).$$

After the substitutions and rearranging terms, we have

$$\begin{aligned}
&\left(\frac{1}{p_i}\left(\frac{1}{2\sigma_i}+\frac{\gamma_i}{2}\right)-\frac{\gamma_i}{2}\right)\|\alpha_i^{(t)}-\alpha_i^\star\|^2+\left(\frac{1}{p_i}-1\right)\big(f_i^*(\alpha_i^{(t)})-f_i^*(\alpha_i^\star)\big)\\
\geq\;& \frac{1}{p_i}\left(\frac{1}{2\sigma_i}+\frac{\gamma_i}{2}\right)\mathbf{E}_t[\|\alpha_i^{(t+1)}-\alpha_i^\star\|^2]+\frac{1}{2p_i\sigma_i}\mathbf{E}_t[\|\alpha_i^{(t+1)}-\alpha_i^{(t)}\|^2]+\frac{1}{p_i}\mathbf{E}_t\big[\big(f_i^*(\alpha_i^{(t+1)})-f_i^*(\alpha_i^\star)\big)\big]\\
&\mathbf{E}_t\big[\langle\alpha_i^\star-\alpha_i^{(t)},\,u_i^{(t+1)}\rangle\big]-\frac{1}{p_i}\mathbf{E}_t\Big[\Big\langle\alpha_i^{(t+1)}-\alpha_i^{(t)},\,u_i^{(t+1)}\Big\rangle\Big].
\end{aligned}$$

Next, we use the assumption $\mathbf{E}_t\big[u_i^{(t+1)}\big]=X_{i:}w^{(t)}$ and the definition of $\widetilde{D}_i(\cdot)$ in (B.2.3) to obtain

$$\begin{aligned}
&\left(\frac{1}{p_i}\left(\frac{1}{2\sigma_i}+\frac{\gamma_i}{2}\right)-\frac{\gamma_i}{2}\right)\|\alpha_i^{(t)}-\alpha_i^\star\|^2-\left(\frac{1}{p_i}-1\right)m\widetilde{D}_i(\alpha_i^{(t)})\\
\geq\;& \frac{1}{p_i}\left(\frac{1}{2\sigma_i}+\frac{\gamma_i}{2}\right)\mathbf{E}_t[\|\alpha_i^{(t+1)}-\alpha_i^\star\|^2]+\frac{1}{2p_i\sigma_i}\mathbf{E}_t[\|\alpha_i^{(t+1)}-\alpha_i^{(t)}\|^2]-\frac{m}{p_i}\mathbf{E}_t\big[\widetilde{D}_i(\alpha_i^{(t+1)})\big]\\
&+\Big\langle\alpha_i^\star-\alpha_i^{(t)},\,X_{i:}\big(w^{(t)}-w^\star\big)\Big\rangle-\frac{1}{p_i}\mathbf{E}_t\Big[\Big\langle\alpha_i^{(t+1)}-\alpha_i^{(t)},\,u_i^{(t+1)}-X_{i:}w^\star\Big\rangle\Big].
\end{aligned}$$

Summing up the above inequality for $i=1,\ldots,m$ gives the desired result (B.2.6). ■

Lemma 7 *Under the same assumptions and setup in Lemma 4, we have*

$$\begin{aligned}
&\sum_{k=1}^K\left(\frac{1}{q_k}\left(\frac{1}{2\tau_k}+\frac{\lambda}{2}\right)-\frac{\lambda}{2}\right)\|w_k^{(t)}-w_k^\star\|^2+\sum_{k=1}^n\left(\frac{1}{q_k}-1\right)\widetilde{P}_k(w_k^{(t)})\\
\geq\;& \sum_{k=1}^n\frac{1}{q_k}\left(\frac{1}{2\tau_k}+\frac{\lambda}{2}\right)\mathbf{E}_t[\|w_k^{(t+1)}-w_k^\star\|^2]+\sum_{k=1}^n\frac{1}{2q_k\tau_k}\mathbf{E}_t[\|w_k^{(t+1)}-w_k^{(t)}\|^2]+\sum_{k=1}^n\frac{1}{q_k}\mathbf{E}_t\big[\widetilde{P}_k(w_k^{(t+1)})\big]\\
&+\frac{1}{m}\big\langle X(w^{(t)}-w^\star),\,\alpha^{(t)}-\alpha^\star\big\rangle+\sum_{k=1}^n\frac{1}{q_k}\mathbf{E}_t\left[\left\langle w_k^{(t+1)}-w_k^{(t)},\,v_k^{(t+1)}-\frac{1}{m}(X_{:k})^T\alpha^\star\right\rangle\right].
\end{aligned}$$

Based on Lemma 6 and Lemma 7, we can prove the following proposition. The proof is very similar to that of Proposition 3, thus we omit the details here.

Proposition 4 *The t-th iteration within the s-th stage of Algorithm 7 guarantees*

$$\begin{aligned}
&\sum_{k=1}^{n}\left(\frac{1}{q_k}-1\right)\widetilde{P}_k(w_k^{(t)})-\sum_{i=1}^{m}\left(\frac{1}{p_i}-1\right)\widetilde{D}_i(\alpha_i^{(t)})\\
&+\sum_{i=1}^{m}\frac{1}{m}\left[\frac{1}{p_i}\left(\frac{1}{2\sigma_i}+\frac{\gamma_i}{2}\right)-\frac{\gamma_i}{2}+\sum_{k=1}^{n}\frac{3\tau_k\|X_{ik}\|^2}{mp_i}\right]\|\alpha_i^{(t)}-\alpha_i^\star\|^2+\sum_{i=1}^{m}\sum_{k=1}^{n}\frac{2\tau_k\|X_{ik}\|^2}{m^2p_i}\|\bar{\alpha}_i^{(s)}-\alpha_i^\star\|^2\\
&+\sum_{k=1}^{n}\left[\frac{1}{q_k}\left(\frac{1}{2\tau_k}+\frac{\lambda}{2}\right)-\frac{\lambda}{2}+\sum_{i=1}^{m}\frac{3\sigma_i\|X_{ik}\|^2}{mq_k}\right]\|w_k^{(t)}-w_k^\star\|^2+\sum_{i=1}^{m}\sum_{k=1}^{n}\frac{2\sigma_i\|X_{ik}\|^2}{mq_k}\|\bar{w}_k^{(s)}-w_k^\star\|^2\\
\geq\ &\sum_{k=1}^{n}\frac{1}{q_k}\mathbf{E}_t\big[\widetilde{P}_k(w_k^{(t+1)})\big]-\sum_{i=1}^{m}\frac{1}{p_i}\mathbf{E}_t\big[\widetilde{D}_i(\alpha_i^{(t+1)})\big]\\
&+\sum_{i=1}^{m}\frac{1}{mp_i}\left(\frac{1}{2\sigma_i}+\frac{\gamma_i}{2}\right)\mathbf{E}_t\big[\|\alpha_i^{(t+1)}-\alpha_i^\star\|^2\big]+\sum_{k=1}^{n}\frac{1}{q_k}\left(\frac{1}{2\tau_k}+\frac{\lambda}{2}\right)\mathbf{E}_t\big[\|w_k^{(t+1)}-w_k^\star\|^2\big]. \quad \text{(B.2.7)}
\end{aligned}$$

Now we proceed to prove Theorem 6.3.2. Let $\theta\in(0,1)$ be a parameter to be determined later, and let Γ and η be two constants such that

$$\Gamma \ \geq\ \max_{i,k}\left\{\frac{1}{p_i}\left(1+\frac{6\Lambda}{\theta q_k\lambda\gamma_i}\right),\ \frac{1}{q_k}\left(1+\frac{6n\Lambda}{\theta p_i m\lambda\gamma_i}\right)\right\}, \quad \text{(B.2.8)}$$

$$\eta \ =\ 1-\frac{1-\theta}{\Gamma}. \quad \text{(B.2.9)}$$

It is easy to check that $\Gamma>1$ and $\eta\in(0,1)$. The choices of σ_i and τ_k in (6.3.14) and (6.3.15) satisfy

$$\frac{1}{p_i}\left(\frac{1}{2}+\frac{1}{2\sigma_i\gamma_i}\right) \ =\ \frac{\Gamma}{2}, \qquad i=1,\ldots,m, \quad \text{(B.2.10)}$$

$$\frac{1}{q_k}\left(\frac{1}{2}+\frac{1}{2\tau_k\lambda}\right) \ =\ \frac{\Gamma}{2}, \qquad k=1,\ldots,n. \quad \text{(B.2.11)}$$

Comparing them with the definition of Γ in (B.2.8), and using the assumption $\Lambda\geq\|X_{ik}\|_F^2\geq\|X_{ik}\|^2$, we get

$$\frac{6\|X_{ik}\|^2}{\theta q_k\lambda\gamma_i}\leq\frac{6\Lambda}{\theta q_k\lambda\gamma_i}\leq\frac{1}{\sigma_i\gamma_i} \qquad\text{and}\qquad \frac{6n\|X_{ik}\|^2}{\theta p_i m\lambda\gamma_i}\leq\frac{6n\Lambda}{\theta p_i m\lambda\gamma_i}\leq\frac{1}{\tau_k\lambda},$$

which implies

$$\frac{3\sigma_i\|X_{ik}\|^2}{q_k}\leq\theta\frac{\lambda}{2} \qquad\text{and}\qquad \frac{3n\tau_k\|X_{ik}\|^2}{mp_i}\leq\theta\frac{\gamma_i}{2},$$

for all $i=1,\ldots,m$ and $k=1,\ldots,n$. Therefore, we have

$$\sum_{k=1}^{n}\frac{3\tau_k\|X_{ik}\|^2}{mp_i}=\frac{1}{n}\sum_{k=1}^{n}\frac{3n\tau_k\|X_{ik}\|^2}{mp_i}\leq\theta\frac{\gamma_i}{2}, \qquad i=1,\ldots,m, \quad \text{(B.2.12)}$$

$$\sum_{i=1}^{m}\frac{3\sigma_i\|X_{ik}\|^2}{mq_k}=\frac{1}{m}\sum_{i=1}^{m}\frac{3\sigma_i\|X_{ik}\|^2}{q_k}\leq\theta\frac{\lambda}{2}, \qquad k=1,\ldots,n. \quad \text{(B.2.13)}$$

Now we consider the inequality (B.2.7), and examine the ratio between the coefficients of $\|\alpha_i^{(t)} - \alpha_i^\star\|^2$ and $\mathbf{E}_t[\|\alpha_i^{(t+1)} - \alpha_i^\star\|^2]$. Using (B.2.12) and (B.2.10), we have

$$\frac{\frac{1}{p_i}\left(\frac{1}{2\sigma_i} + \frac{\gamma_i}{2}\right) - \frac{\gamma_i}{2} + \sum_{k=1}^n \frac{3\tau_k \|X_{ik}\|^2}{mp_i}}{\frac{1}{p_i}\left(\frac{1}{2\sigma_i} + \frac{\gamma_i}{2}\right)} \leq 1 - \frac{(1-\theta)\frac{\gamma_i}{2}}{\frac{1}{p_i}\left(\frac{1}{2\sigma_i} + \frac{\gamma_i}{2}\right)} = 1 - \frac{1-\theta}{\Gamma} = \eta. \tag{B.2.14}$$

Similarly, the ratio between the coefficients of $\|w_k^{(t)} - w_k^\star\|^2$ and $\mathbf{E}_t[\|w_k^{(t+1)} - w_k^\star\|^2]$ can be bounded using (B.2.13) and (B.2.11):

$$\frac{\frac{1}{q_k}\left(\frac{1}{2\tau_k} + \frac{\lambda}{2}\right) - \frac{\lambda}{2} + \sum_{i=1}^m \frac{3\sigma_i \|X_{ik}\|^2}{mq_k}}{\frac{1}{q_k}\left(\frac{1}{2\tau_k} + \frac{\lambda}{2}\right)} \leq 1 - \frac{(1-\theta)\frac{\lambda}{2}}{\frac{1}{q_k}\left(\frac{1}{2\tau_k} + \frac{\lambda}{2}\right)} = 1 - \frac{1-\theta}{\Gamma} = \eta. \tag{B.2.15}$$

In addition, the ratio between the coefficients of $\|\bar{\alpha}_i^{(s)} - \alpha_i^\star\|^2$ and $\mathbf{E}_t[\|\alpha_i^{(t+1)} - \alpha_i^\star\|^2]$ and that of $\|\bar{w}_k^{(s)} - w_k^\star\|^2$ and $\mathbf{E}_t[\|w_k^{(t+1)} - w_k^\star\|^2]$ can be bounded as

$$\frac{\sum_{k=1}^n \frac{2\tau_k \|X_{ik}\|^2}{mp_i}}{\frac{1}{p_i}\left(\frac{1}{2\sigma_i} + \frac{\gamma_i}{2}\right)} \leq \frac{(2/3)\theta\frac{\gamma_i}{2}}{\frac{1}{p_i}\left(\frac{1}{2\sigma_i} + \frac{\gamma_i}{2}\right)} = \frac{(2/3)\theta}{\Gamma} = \frac{2\theta(1-\eta)}{3(1-\theta)}, \tag{B.2.16}$$

$$\frac{\sum_{i=1}^m \frac{2\sigma_i \|X_{ik}\|^2}{mq_k}}{\frac{1}{q_k}\left(\frac{1}{2\tau_k} + \frac{\lambda}{2}\right)} \leq \frac{(2/3)\theta\frac{\lambda}{2}}{\frac{1}{q_k}\left(\frac{1}{2\tau_k} + \frac{\lambda}{2}\right)} = \frac{(2/3)\theta}{\Gamma} = \frac{2\theta(1-\eta)}{3(1-\theta)}. \tag{B.2.17}$$

Also, the ratios between the coefficients of $\widetilde{P}_k(w_k^{(t)})$ and $\mathbf{E}_t\big[\widetilde{P}_k(w_k^{(t+1)})\big]$ is $1 - q_k$, and that of $\widetilde{D}_k(\alpha_i^{(t)})$ and $\mathbf{E}_t\big[\widetilde{D}_i(\alpha_i^{(t+1)})\big]$ is $1 - p_i$. From the definition of Γ and η in (B.2.8) and (B.2.9), we have

$$1 - p_i \leq \eta \quad \text{for} \quad i = 1, \ldots, m, \quad \text{and} \quad 1 - q_k \leq \eta \quad \text{for} \quad k = 1, \ldots, n. \tag{B.2.18}$$

Using the relations in (B.2.10) and (B.2.11) and the inequalities (B.2.14), (B.2.15), (B.2.16), (B.2.17) and (B.2.18), we conclude that the inequality (B.2.7) implies

$$\begin{aligned}
&\eta\left(\sum_{k=1}^n \frac{1}{q_k}\widetilde{P}_k(w_k^{(t)}) - \sum_{i=1}^m \frac{1}{p_i}\widetilde{D}_i(\alpha_i^{(t)})\right) + \eta\left(\sum_{i=1}^m \frac{\Gamma\gamma_i}{2m}\|\alpha_i^{(t)} - \alpha_i^\star\|^2 + \sum_{k=1}^n \frac{\Gamma\lambda}{2}\|w_k^{(t)} - w_k^\star\|^2\right) \\
&+\frac{2\theta(1-\eta)}{3(1-\theta)}\left(\sum_{i=1}^m \frac{\Gamma\gamma_i}{2m}\|\bar{\alpha}_i^{(s)} - \alpha_i^\star\|^2 \sum_{k=1}^n \frac{\Gamma\lambda}{2}\|\bar{w}_k^{(s)} - w_k^\star\|^2\right) \\
\geq\ & \sum_{k=1}^n \frac{1}{q_k}\mathbf{E}_t\big[\widetilde{P}_k(w_k^{(t+1)})\big] - \sum_{i=1}^m \frac{1}{p_i}\mathbf{E}_t\big[\widetilde{D}_i(\alpha_i^{(t+1)})\big] \\
&+\sum_{i=1}^m \frac{\Gamma\gamma_i}{2m}\mathbf{E}_t[\|\alpha_i^{(t+1)} - \alpha_i^\star\|^2] + \sum_{k=1}^n \frac{\Gamma\lambda}{2}\mathbf{E}_t[\|w_k^{(t+1)} - w_k^\star\|^2],
\end{aligned}$$

which is equivalent to

$$\begin{aligned}
&\eta\left(\sum_{k=1}^{n}\frac{1}{q_k}\widetilde{P}_k(w_k^{(t)})-\sum_{i=1}^{m}\frac{1}{p_i}\widetilde{D}_i(\alpha_i^{(t)})+\frac{\Gamma\lambda}{2}\|w^{(t)}-w^\star\|^2+\frac{1}{m}\sum_{i=1}^{m}\frac{\Gamma\gamma_i}{2}\|\alpha_i^{(t)}-\alpha_i^\star\|^2\right)\\
&+\frac{2\theta(1-\eta)}{3(1-\theta)}\left(\frac{\Gamma\lambda}{2}\|\bar{w}^{(s)}-w^\star\|^2+\frac{1}{m}\sum_{i=1}^{m}\frac{\Gamma\gamma_i}{2}\|\bar{\alpha}_i^{(s)}-\alpha_i^\star\|^2\right) \qquad \text{(B.2.19)}\\
\geq\ &\mathbf{E}_t\left[\sum_{k=1}^{n}\frac{1}{q_k}\widetilde{P}_k(w_k^{(t+1)})-\sum_{i=1}^{m}\frac{1}{p_i}\widetilde{D}_i(\alpha_i^{(t+1)})+\frac{\Gamma\lambda}{2}\|w^{(t+1)}-w^\star\|^2+\frac{1}{m}\sum_{i=1}^{m}\frac{\Gamma\gamma_i}{2}\|\alpha_i^{(t+1)}-\alpha_i^\star\|^2\right].
\end{aligned}$$

To simplify further derivation, we define

$$\begin{aligned}
\Delta^{(t)} &= \sum_{k=1}^{n}\frac{1}{q_k}\widetilde{P}_k(w_k^{(t)})-\sum_{i=1}^{m}\frac{1}{p_i}\widetilde{D}_i(\alpha_i^{(t)})+\frac{\Gamma\lambda}{2}\|w^{(t)}-w^\star\|^2+\frac{1}{m}\sum_{i=1}^{m}\frac{\Gamma\gamma_i}{2}\|\alpha_i^{(t)}-\alpha_i^\star\|^2,\\
\bar{\Delta}^{(s)} &= \sum_{k=1}^{n}\frac{1}{q_k}\widetilde{P}_k(\bar{w}_k^{(s)})-\sum_{i=1}^{m}\frac{1}{p_i}\widetilde{D}_i(\bar{\alpha}_i^{(s)})+\frac{\Gamma\lambda}{2}\|\bar{w}^{(s)}-w^\star\|^2+\frac{1}{m}\sum_{i=1}^{m}\frac{\Gamma\gamma_i}{2}\|\bar{\alpha}_i^{(s)}-\alpha_i^\star\|^2.
\end{aligned}$$

Using the facts that $\widetilde{P}_k(\bar{w}_k^{(s)})\geq 0$ and $-\widetilde{D}_i(\bar{\alpha}_i^{(s)})\geq 0$, the inequality (B.2.19) implies

$$\frac{2\theta(1-\eta)}{3(1-\theta)}\bar{\Delta}^{(s)}+\eta\mathbf{E}\big[\Delta^{(t)}\big]\ \geq\ \mathbf{E}\big[\Delta^{(t+1)}\big],$$

where the expectation is taken with respect to all randomness in the s-th stage, that is, the random variables $\{(j^{(0)},l^{(0)}),(j^{(1)},l^{(1)}),\ldots,(j^{(M-1)},l^{(M-1)})\}$. Next we choose $\theta=1/3$ and follow the same arguments as in the proof for Theorem 6.3.1 to obtain $\mathbf{E}\big[\Delta^{(M)}\big]\ \leq\ \frac{2}{3}\Delta^{(0)}$, provided $M\geq\log(3)\Gamma$. This further implies

$$\mathbf{E}\big[\bar{\Delta}^{(s)}\big]\ \leq\ \left(\frac{2}{3}\right)^s\bar{\Delta}^{(0)}. \qquad \text{(B.2.20)}$$

From the definition of Γ in (B.2.8), we have $\frac{1}{q_k}<\Gamma$ for $k=1,\ldots,n$ and $\frac{1}{p_i}<\Gamma$ for $i=1,\ldots,m$. Therefore,

$$\begin{aligned}
\bar{\Delta}^{(0)} &\leq \Gamma\left(\sum_{k=1}^{n}\widetilde{P}_k(\bar{w}_k^{(0)})-\sum_{i=1}^{m}\widetilde{D}_i(\bar{\alpha}_i^{(0)})+\frac{\lambda}{2}\|\bar{w}^{(0)}-w^\star\|^2+\frac{1}{m}\sum_{i=1}^{m}\frac{\gamma_i}{2}\|\bar{\alpha}_i^{(0)}-\alpha_i^\star\|^2\right)\\
&\leq 2\Gamma\left(\sum_{k=1}^{n}\widetilde{P}_k(\bar{w}_k^{(0)})-\sum_{i=1}^{m}\widetilde{D}_i(\bar{\alpha}_i^{(0)})\right)\\
&\leq 2\Gamma\left(P(\bar{w}^{(0)})-D(\bar{\alpha}^{(0)})\right), \qquad \text{(B.2.21)}
\end{aligned}$$

where the second inequality used (B.2.4) and the last inequality used (B.2.5). On the other hand, we can also lower bound $\bar{\Delta}^{(s)}$ using $P(\bar{w}^{(s)})-D(|)|$. To this end, we notice that with $\theta=1/3$,

$$\Gamma\ \geq\ \max_{i,k}\left\{\frac{1}{p_i}\left(1+\frac{18\Lambda}{q_k\lambda\gamma_i}\right),\ \frac{1}{q_k}\left(1+\frac{18n\Lambda}{p_im\lambda\gamma_i}\right)\right\}\ \geq\ \max_{i,k}\left\{\frac{18\Lambda}{p_iq_k\lambda\gamma_i},\ \frac{18n\Lambda}{p_iq_km\lambda\gamma_i}\right\}.$$

Noticing that $\max_k\{1/q_k\} \geq n$ and $n\Lambda \geq \|X_{i:}\|_F^2$ for all $i = 1, \ldots, m$, we have

$$\Gamma \geq \max_{i,k}\left\{\frac{18\Lambda}{q_k\lambda\gamma_i}\right\} \geq \max_i\left\{\frac{18n\Lambda}{\lambda\gamma_i}\right\} \geq \frac{18}{m\lambda}\sum_{i=1}^m \frac{n\Lambda}{\gamma_i} \geq \frac{18}{m\lambda}\sum_{i=1}^m \frac{\|X_{i:}\|_F^2}{\gamma_i} \geq \frac{18}{m\lambda}\sum_{i=1}^m \frac{\|X_{i:}\|^2}{\gamma_i}.$$

Moreover, since $\Gamma \geq \max_k\left\{\frac{18\Lambda}{p_iq_k\lambda\gamma_i}\right\} \geq \frac{18n\Lambda}{p_i\lambda\gamma_i}$ for all i and $mn\Lambda \geq \|X\|_F^2$, we have

$$\begin{aligned}
\frac{1}{m}\sum_{i=1}^m \Gamma\gamma_i\|\bar{\alpha}_i^{(s)} - \alpha_i^\star\|^2 &\geq \frac{1}{m}\sum_{i=1}^m \frac{18n\Lambda}{p_i\lambda\gamma_i}\gamma_i\|\bar{\alpha}_i^{(s)} - \alpha_i^\star\|^2 = \frac{18mn\Lambda}{m^2\lambda}\sum_{i=1}^m \frac{\|\bar{\alpha}_i^{(s)} - \alpha_i^\star\|^2}{p_i} \\
&\geq \frac{18\|X\|_F^2}{m^2\lambda}\left(\sum_{i=1}^m \|\bar{\alpha}_i^{(s)} - \alpha_i^\star\|\right)^2 \\
&\geq \frac{18\|X\|^2}{m^2\lambda}\sum_{i=1}^m \|\bar{\alpha}_i^{(s)} - \alpha_i^\star\|^2 = \frac{18\|X\|^2}{m^2\lambda}\|\,|-|\,\alpha^\star\|^2.
\end{aligned}$$

Therefore, from the definition of $\bar{\Delta}^{(s)}$,

$$\begin{aligned}
\bar{\Delta}^{(s)} &= \sum_{k=1}^n \frac{1}{q_k}\widetilde{P}_k(\bar{w}_k^{(s)}) - \sum_{i=1}^m \frac{1}{p_i}\widetilde{D}_i(\bar{\alpha}_i^{(s)}) + \frac{\Gamma\lambda}{2}\|\bar{w}^{(s)} - w^\star\|^2 + \frac{1}{m}\sum_{i=1}^m \frac{\Gamma\gamma_i}{2}\|\bar{\alpha}_i^{(s)} - \alpha_i^\star\|^2 \\
&\geq \sum_{k=1}^n \widetilde{P}_k(\bar{w}_k^{(s)}) - \sum_{i=1}^m \widetilde{D}_i(\bar{\alpha}_i^{(s)}) + \left(\frac{18}{m}\sum_{i=1}^m \frac{\|X_{i:}\|^2}{\gamma_i}\right)\|\bar{w}^{(s)} - w^\star\|^2 + \frac{18\|X\|^2}{m^2\lambda}\|\,|-|\,\alpha^\star\|^2 \\
&= L(\bar{w}^{(s)}, \alpha^\star) - L(w^\star, |)| + \left(\frac{18}{m}\sum_{i=1}^m \frac{\|X_{i:}\|^2}{\gamma_i}\right)\|\bar{w}^{(s)} - w^\star\|^2 + \frac{18\|X\|^2}{m^2\lambda}\|\,|-|\,\alpha^\star\|^2 \\
&\geq P(\bar{w}^{(s)}) - D(|)|\,, \qquad \text{(B.2.22)}
\end{aligned}$$

where the last inequality is due to Lemma 5. Combining (B.2.20), (B.2.21) and (B.2.22) gives the desired result:

$$\mathbf{E}\left[P(\bar{w}^{(s)}) - D(|)|\right] \leq \left(\frac{2}{3}\right)^s 2\Gamma\left(P(\bar{w}^{(0)}) - D(\bar{\alpha}^{(0)})\right).$$

B.3 Proof of Theorem 6.4.1

To facilitate the analysis of DSCOVR-SAGA in Algorithm 8, we define two sequences of matrices recursively. The first is $\{W^{(t)}\}_{t\geq 0}$, where each $W^{(t)} \in \mathbf{R}^{m\times d}$. They are partitioned into $m \times n$ blocks, and we denote each block as $W_{ik}^{(t)} \in \mathbf{R}^{1\times d_k}$. The recursive updates for $W^{(t)}$ are as follows:

$$\begin{aligned}
W^{(0)} &= \mathbf{1}_m \otimes \left(w^{(0)}\right)^T, \\
W_{ik}^{(t+1)} &= \begin{cases} \left(w_l^{(t)}\right)^T & \text{if } i = j \text{ and } k = l, \\ W_{ik}^{(t)} & \text{otherwise,} \end{cases} \qquad t = 0, 1, 2, \ldots, \qquad \text{(B.3.1)}
\end{aligned}$$

where $\mathbf{1}_m$ denotes the vector of all ones in $\mathbf{R}^m$. and $\otimes$ denotes the Kronecker product of two matrices. The second sequence is $\{A^{(t)}\}_{t\geq 0}$, where each $A^{(t)} \in \mathbf{R}^{N\times n}$. They are partitioned into $m \times n$ blocks, and we denote each block as $A_{ik}^{(t)} \in \mathbf{R}^{N_i\times 1}$. The recursive updates for $A^{(t)}$ are as follows:

$$\begin{aligned} A^{(0)} &= \alpha^{(0)} \otimes \mathbf{1}_n^T, \\ A_{ik}^{(t+1)} &= \begin{cases} \alpha_j^{(t)} & \text{if } i = j \text{ and } k = l, \\ A_{ik}^{(t)} & \text{otherwise,} \end{cases} \qquad t = 0, 1, 2, \ldots. \end{aligned} \tag{B.3.2}$$

The matrices $W^{(t)}$ and $A^{(t)}$ consist of most recent values of the primal and dual block coordinates, updated at different times, up to time t.

Notice that in Algorithm 8, the matrices $U^{(t)} \in \mathbf{R}^{N\times n}$ follow the same partitioning as the matrices $A^{(t)}$, and the matrices $V^{(t)} \in \mathbf{R}^{m\times d}$ follow the same partitioning as the matrices $W^{(t)}$. According to the updates of $U^{(t)}$, $V^{(t)}$, $\bar{u}^{(t)}$ and $\bar{v}^{(t)}$ in Algorithm 8, we have for each $t \geq 0$,

$$\begin{aligned} U_{ik}^{(t)} &= X_{ik}\big(W_{ik}^{(t)}\big)^T, \quad i = 1, \ldots, m, \quad k = 1, \ldots, n, \\ V_{ik}^{(t)} &= \frac{1}{m}\big(A_{ik}^{(t)}\big)^T X_{ik}, \quad i = 1, \ldots, m, \quad k = 1, \ldots, n. \end{aligned}$$

Proposition 5 *Suppose Assumption 2 holds. The t-th iteration of Algorithm 8 guarantees*

$$\begin{aligned} &\sum_{i=1}^m \frac{1}{m}\left[\frac{1}{p_i}\left(\frac{1}{2\sigma_i} + \gamma_i\right) - \gamma_i + \sum_{k=1}^n \frac{3\tau_k\|X_{ik}\|^2}{mp_i}\right]\|\alpha_i^{(t)} - \alpha_i^\star\|^2 + \sum_{i=1}^m\sum_{k=1}^n \frac{2\tau_k\|X_{ik}\|^2}{m^2p_i}\|A_{ik}^{(t)} - \alpha_i^\star\|^2 \\ &+ \sum_{k=1}^n \left[\frac{1}{q_k}\left(\frac{1}{2\tau_k} + \lambda\right) - \lambda + \sum_{i=1}^m \frac{3\sigma_i\|X_{ik}\|^2}{mq_k}\right]\|w_k^{(t)} - w_k^\star\|^2 + \sum_{i=1}^m\sum_{k=1}^n \frac{2\sigma_i\|X_{ik}\|^2}{mq_k}\|\big(W_{ik}^{(t)}\big)^T - w_k^\star\|^2 \\ \geq\ & \sum_{i=1}^m \frac{1}{mp_i}\left(\frac{1}{2\sigma_i} + \gamma_i\right)\mathbf{E}_t\big[\|\alpha_i^{(t+1)} - \alpha_i^\star\|^2\big] + \sum_{k=1}^n \frac{1}{q_k}\left(\frac{1}{2\tau_k} + \lambda\right)\mathbf{E}_t\big[\|w_k^{(t+1)} - w_k^\star\|^2\big] \end{aligned} \tag{B.3.3}$$

Proof: The main differences between Algorithm 7 and Algorithm 8 are the definitions of $u_j^{(t+1)}$ and $v_l^{(t+1)}$. We start with the inequality (B.1.11) and revisit the bounds for the following two quantities:

$$\mathbf{E}_t\big[\|u_i^{(t+1)} - X_{i:}w^{(t)}\|^2\big] \qquad \text{and} \qquad \mathbf{E}_t\left[\left\|v_k^{(t+1)} - \frac{1}{m}(X_{:k})^T\alpha^{(t)}\right\|^2\right].$$

For Algorithm 8, we have

$$\begin{aligned} u_i^{(t+1)} &= \bar{u}_i^{(t)} - \frac{1}{q_l}U_{il}^{(t)} + \frac{1}{q_l}X_{il}w_l^{(t)}, \qquad i = 1, \ldots, m, \\ v_k^{(t+1)} &= \bar{v}_k^{(t)} - \frac{1}{p_j}(V_{jk}^{(t)})^T + \frac{1}{p_j}\frac{1}{m}(X_{jk})^T\alpha_j^{(t)}, \qquad k = 1, \ldots, n. \end{aligned}$$

We can apply the reasoning in (6.4.5) and (6.4.6) to every block coordinate and obtain

$$\begin{aligned} \mathbf{E}_t\big[u_i^{(t+1)}\big] &= X_{i:}w^{(t)}, \qquad i = 1, \ldots, m, \\ \mathbf{E}_t\big[v_k^{(t+1)}\big] &= \frac{1}{m}(X_{:k})^T\alpha^{(t)}, \qquad k = 1, \ldots, n. \end{aligned}$$

Therefore they satisfy the assumptions in Lemma 3 and Lemma 4, respectively. Moreover, following similar arguments as in (B.1.12) and (B.1.13), we have

$$\begin{aligned}
\mathbf{E}_t\big[\|u_i^{(t+1)} - X_{i:}w^{(t)}\|^2\big] &\leq \sum_{k=1}^{n} \frac{2\|X_{ik}\|^2}{q_k}\left(\Big\|\big(W_{ik}^{(t)}\big)^T - w_k^\star\Big\|^2 + \|w_k^{(t)} - w_k^\star\|^2\right), \\
\mathbf{E}_t\left[\Big\|v_k^{(t+1)} - \frac{1}{m}(X_{:k})^T\alpha^{(t)}\Big\|^2\right] &\leq \sum_{i=1}^{m} \frac{2\|X_{ik}\|^2}{m^2 p_i}\left(\Big\|\big(A_{ik}^{(t)}\big)^T - \alpha_i^\star\Big\|^2 + \|\alpha_i^{(t)} - \alpha_i^\star\|^2\right).
\end{aligned}$$

The rest of the proof are the same as in the proof of Proposition 3. ■

Now we are ready to prove Theorem 6.4.1. By the definition of $W^{(t)}$ in (B.3.1) and $A^{(t)}$ in (B.3.2), we have

$$\begin{aligned}
\mathbf{E}_t\left[\Big\|\big(W_{ik}^{(t+1)}\big)^T - w_k^\star\Big\|^2\right] &= p_i q_k\big\|w_k^{(t)} - w_k^\star\big\|^2 + (1 - p_i q_k)\Big\|\big(W_{ik}^{(t)}\big)^T - w_k^\star\Big\|^2, \quad \text{(B.3.4)} \\
\mathbf{E}_t\left[\big\|A_{ik}^{(t+1)} - \alpha_i^\star\big\|^2\right] &= p_i q_k\big\|\alpha_i^{(t)} - \alpha_i^\star\big\|^2 + (1 - p_i q_k)\big\|A_{ik}^{(t)} - \alpha_i^\star\big\|^2. \quad \text{(B.3.5)}
\end{aligned}$$

For all $i = 1, \ldots, m$ and $k = 1, \ldots, n$, let

$$\xi_{ik} = \frac{3\sigma_i\|X_{ik}\|^2}{m p_i q_k^2} \qquad \text{and} \qquad \zeta_{ik} = \frac{3\tau_k\|X_{ik}\|^2}{m^2 p_i^2 q_k}. \quad \text{(B.3.6)}$$

We multiply (B.3.4) by ξ_{ik} and (B.3.5) by ζ_{ik} and add them to (B.3.3) to obtain

$$\begin{aligned}
&\sum_{i=1}^{m}\frac{1}{m}\left[\frac{1}{p_i}\left(\frac{1}{2\sigma_i} + \gamma_i\right) - \gamma_i + \sum_{k=1}^{n}\frac{6\tau_k\|X_{ik}\|^2}{m p_i}\right]\|\alpha_i^{(t)} - \alpha_i^\star\|^2 \\
&+ \sum_{k=1}^{n}\left[\frac{1}{q_k}\left(\frac{1}{2\tau_k} + \lambda\right) - \lambda + \sum_{i=1}^{m}\frac{6\sigma_i\|X_{ik}\|^2}{m q_k}\right]\|w_k^{(t)} - w_k^\star\|^2 \\
&+ \sum_{i=1}^{m}\sum_{k=1}^{n}\left(1 - \frac{1}{3}p_i q_k\right)\zeta_{ik}\big\|A_{ik}^{(t)} - \alpha_i^\star\big\|^2 + \sum_{i=1}^{m}\sum_{k=1}^{n}\left(1 - \frac{1}{3}p_i q_k\right)\xi_{ik}\Big\|\big(W_{ik}^{(t)}\big)^T - w_k^\star\Big\|^2 \\
\geq\ &\sum_{i=1}^{m}\frac{1}{m p_i}\left(\frac{1}{2\sigma_i} + \gamma_i\right)\mathbf{E}_t\big[\|\alpha_i^{(t+1)} - \alpha_i^\star\|^2\big] + \sum_{k=1}^{n}\frac{1}{q_k}\left(\frac{1}{2\tau_k} + \lambda\right)\mathbf{E}_t\big[\|w_k^{(t+1)} - w_k^\star\|^2\big] \\
&+ \sum_{i=1}^{m}\sum_{k=1}^{n}\zeta_{ik}\mathbf{E}_t\big[\big\|A_{ik}^{(t+1)} - \alpha_i^\star\big\|^2\big] + \sum_{i=1}^{m}\sum_{k=1}^{n}\xi_{ik}\mathbf{E}_t\big[\Big\|\big(W_{ik}^{(t+1)}\big)^T - w_k^\star\Big\|^2\big]. \quad \text{(B.3.7)}
\end{aligned}$$

Let $\theta \in (0, 1)$ be a parameter to be determined later, and Γ be a constant such that

$$\Gamma \geq \max_{i,k}\left\{\frac{1}{p_i}\left(1 + \frac{3\|X_{ik}\|^2}{2\theta q_k\lambda\gamma_i}\right),\ \frac{1}{q_k}\left(1 + \frac{3n\|X_{ik}\|^2}{2\theta p_i m\lambda\gamma_i}\right),\ \frac{1}{p_i q_k}\right\}. \quad \text{(B.3.8)}$$

The choices of σ_i in (6.4.8) and τ_k in (6.4.9) satisfy

$$\frac{1}{p_i}\left(1 + \frac{1}{2\sigma_i\gamma_i}\right) = \frac{1}{q_k}\left(1 + \frac{1}{2\tau_k\lambda}\right) = \Gamma. \quad \text{(B.3.9)}$$

Comparing the above equality with the definition of Γ in (B.3.8), we have

$$\frac{3\|X_{ik}\|^2}{2\theta q_k \lambda \gamma_i} \le \frac{1}{2\sigma_i \gamma_i} \qquad \text{and} \qquad \frac{3n\|X_{ik}\|^2}{2\theta p_i m \lambda \gamma_i} \le \frac{1}{2\tau_k \lambda},$$

which implies that

$$\frac{6\sigma_i\|X_{ik}\|^2}{q_k} \le 2\theta\lambda \qquad \text{and} \qquad \frac{6n\tau_k\|X_{ik}\|^2}{mp_i} \le 2\theta\gamma_i \tag{B.3.10}$$

hold for all $i = 1, \ldots, m$ and $k = 1, \ldots, n$. Therefore, we have

$$\sum_{k=1}^{n} \frac{6\tau_k\|X_{ik}\|^2}{mp_i} = \frac{1}{n}\sum_{k=1}^{n} \frac{6n\tau_k\|X_{ik}\|^2}{mp_i} \le 2\theta\gamma_i, \qquad i = 1, \ldots, m, \tag{B.3.11}$$

$$\sum_{i=1}^{m} \frac{6\sigma_i\|X_{ik}\|^2}{mq_k} = \frac{1}{m}\sum_{i=1}^{m} \frac{6\sigma_i\|X_{ik}\|^2}{q_k} \le 2\theta\lambda, \qquad k = 1, \ldots, n. \tag{B.3.12}$$

Now we consider the inequality (B.3.7), and examine the ratio between the coefficients of $\|\alpha_i^{(t)} - \alpha_i^\star\|^2$ and $\mathbf{E}_t[\|\alpha_i^{(t+1)} - \alpha_i^\star\|^2]$. Using (B.3.11) and (B.3.9), we have

$$\frac{\frac{1}{p_i}\left(\frac{1}{2\sigma_i} + \gamma_i\right) - \gamma_i + \sum_{k=1}^{n} \frac{6\tau_k\|X_{ik}\|^2}{mp_i}}{\frac{1}{p_i}\left(\frac{1}{2\sigma_i} + \gamma_i\right)} \le 1 - \frac{(1-2\theta)\gamma_i}{\frac{1}{p_i}\left(\frac{1}{2\sigma_i} + \gamma_i\right)} = 1 - \frac{1-2\theta}{\Gamma}. \tag{B.3.13}$$

Similarly, the ratio between the coefficients of $\|w_k^{(t)} - w_k^\star\|^2$ and $\mathbf{E}_t[\|w_k^{(t+1)} - w_k^\star\|^2]$ can be bounded using (B.3.12) and (B.3.9):

$$\frac{\frac{1}{q_k}\left(\frac{1}{2\tau_k} + \lambda\right) - \lambda + \sum_{i=1}^{m} \frac{6\sigma_i\|X_{ik}\|^2}{mq_k}}{\frac{1}{q_k}\left(\frac{1}{2\tau_k} + \lambda\right)} \le 1 - \frac{(1-2\theta)\lambda}{\frac{1}{q_k}\left(\frac{1}{2\tau_k} + \lambda\right)} = 1 - \frac{1-2\theta}{\Gamma}. \tag{B.3.14}$$

We notice that in (B.3.7), the ratios between the coefficients of $\zeta_{ik}\left\|A_{ik}^{(t)} - \alpha_i^\star\right\|^2$ and $\zeta_{ik}\mathbf{E}_t\left[\left\|A_{ik}^{(t+1)} - \alpha_i^\star\right\|^2\right]$, as well as and that of $\xi_{ik}\left\|\left(W_{ik}^{(t)}\right)^T - w_k^\star\right\|^2$ and $\xi_{ik}\mathbf{E}_t\left[\left\|\left(W_{ik}^{(t+1)}\right)^T - w_k^\star\right\|^2\right]$, are all $1 - \frac{1}{3}p_i q_k$. By definition of Γ in (B.3.8), we have

$$1 - \frac{1}{3}p_i q_k \;\le\; 1 - \frac{1}{3\Gamma}, \qquad i = 1, \ldots, m, \quad k = 1, \ldots, n. \tag{B.3.15}$$

We choose $\theta = 1/3$ so that the ratios in (B.3.13) and (B.3.14) have the same bound $1 - \frac{1}{3\Gamma}$.

Therefore, it follows from inequality (B.3.7) that

$$\begin{aligned}
&\sum_{i=1}^{m}\frac{\Gamma\gamma_i}{m}\mathbf{E}_t\big[\|\alpha_i^{(t+1)}-\alpha_i^\star\|^2\big]+\sum_{k=1}^{n}\Gamma\lambda\mathbf{E}_t\big[\|w_k^{(t+1)}-w_k^\star\|^2\big]\\
&+\sum_{i=1}^{m}\sum_{k=1}^{n}\zeta_{ik}\mathbf{E}_t\big[\big\|A_{ik}^{(t+1)}-\alpha_i^\star\big\|^2\big]+\sum_{i=1}^{m}\sum_{k=1}^{n}\xi_{ik}\mathbf{E}_t\big[\big\|\big(W_{ik}^{(t+1)}\big)^T-w_k^\star\big\|^2\big].\\
\leq\ &\left(1-\frac{1}{3\Gamma}\right)\left(\sum_{i=1}^{m}\frac{\Gamma\gamma_i}{m}\|\alpha_i^{(t)}-\alpha_i^\star\|^2+\sum_{k=1}^{n}\Gamma\lambda\|w_k^{(t)}-w_k^\star\|^2\right.\\
&\left.+\sum_{i=1}^{m}\sum_{k=1}^{n}\zeta_{ik}\big\|A_{ik}^{(t)}-\alpha_i^\star\big\|^2+\sum_{i=1}^{m}\sum_{k=1}^{n}\xi_{ik}\big\|\big(W_{ik}^{(t)}\big)^T-w_k^\star\big\|^2\right). \quad \text{(B.3.16)}
\end{aligned}$$

Let's define

$$\Delta^{(t)}=\lambda\|w^{(t)}-w^\star\|^2+\frac{1}{m}\sum_{i=1}^{m}\gamma_i\|\alpha_i^{(t)}-\alpha_i^\star\|^2+\sum_{i=1}^{m}\sum_{k=1}^{n}\frac{\zeta_{ik}}{\Gamma}\big\|A_{ik}^{(t)}-\alpha_i^\star\big\|^2+\sum_{i=1}^{m}\sum_{k=1}^{n}\frac{\xi_{ik}}{\Gamma}\big\|\big(W_{ik}^{(t)}\big)^T-w_k^\star\big\|^2.$$

Then (B.3.16) implies

$$\mathbf{E}\left[\Delta^{(t)}\right]\leq\left(1-\frac{1}{3\Gamma}\right)^t\Delta^{(0)}, \quad \text{(B.3.17)}$$

where the expectation is taken with respect to all random variables generated by Algorithm 8 up to iteration t.

By the definition of ξ_{ik} in (B.3.6), we have

$$\frac{\xi_{ik}}{\Gamma}=\frac{3\sigma_i\|X_{ik}\|^2}{mp_iq_k^2}\frac{1}{\Gamma}\leq\frac{\theta\lambda}{mp_iq_k}\frac{1}{\Gamma}\leq\frac{\theta\lambda}{m}=\frac{\lambda}{3m},$$

where the first inequality is due to (B.3.10) and the second inequality is due to the relation $\Gamma\geq\frac{1}{p_iq_k}$ from the definition of Γ in (B.3.8). Similarly, we have

$$\frac{\zeta_{ik}}{\Gamma}=\frac{3\tau_k\|X_{ik}\|^2}{m^2p_i^2q_k}\frac{1}{\Gamma}\leq\frac{\theta\gamma_i}{mnp_iq_k}\frac{1}{\Gamma}\leq\frac{\theta\gamma_i}{3mn}=\frac{\gamma_i}{3mn}.$$

Moreover, by the construction in (B.3.1) and (B.3.2), we have for $t=0$,

$$\begin{aligned}
A_{ik}^{(0)}&=\alpha_i^{(0)},\quad\text{for}\quad k=1,\dots,n\quad\text{and}\quad i=1,\dots,m,\\
\big(W_{ik}^{(0)}\big)^T&=w_k^{(0)},\quad\text{for}\quad i=1,\dots,m\quad\text{and}\quad k=1,\dots,n.
\end{aligned}$$

Therefore, the last two terms in the definition of $\Delta^{(0)}$ can be bounded as

$$\begin{aligned}
&\sum_{i=1}^{m}\sum_{k=1}^{n}\frac{\zeta_{ik}}{\Gamma}\big\|A_{ik}^{(0)}-\alpha_i^\star\big\|^2+\sum_{i=1}^{m}\sum_{k=1}^{n}\frac{\xi_{ik}}{\Gamma}\big\|\big(W_{ik}^{(0)}\big)^T-w_k^\star\big\|^2\\
\leq\ &\sum_{i=1}^{m}\sum_{k=1}^{n}\frac{\gamma_i}{3mn}\big\|\alpha_i^{(0)}-\alpha_i^\star\big\|^2+\sum_{i=1}^{m}\sum_{k=1}^{n}\frac{\lambda}{3m}\big\|w_k^{(0)}-w_k^\star\big\|^2\\
=\ &\frac{1}{3m}\sum_{i=1}^{m}\gamma_i\big\|\alpha_i^{(0)}-\alpha_i^\star\big\|^2+\frac{\lambda}{3}\big\|w^{(0)}-w^\star\big\|^2,
\end{aligned}$$

which implies

$$\Delta^{(0)} \leq \frac{4}{3}\left(\lambda\left\|w^{(0)} - w^\star\right\|^2 + \frac{1}{m}\sum_{i=1}^m \gamma_i \left\|\alpha_i^{(0)} - \alpha_i^\star\right\|^2\right).$$

Finally, combining with (B.3.17), we have

$$\mathbf{E}\left[\Delta^{(t)}\right] \leq \left(1 - \frac{1}{3\Gamma}\right)^t \frac{4}{3}\left(\lambda\left\|w^{(0)} - w^\star\right\|^2 + \frac{1}{m}\sum_{i=1}^m \gamma_i \left\|\alpha_i^{(0)} - \alpha_i^\star\right\|^2\right),$$

which further implies the desired result.

B.4 Proof of Theorem 6.5.1

To simplify the presentation, we present the proof for the case $\gamma_i = \gamma$ for all $i = 1, \ldots, m$. It is straightforward to generalize to the case where the γ_i's are different.

Lemma 8 *Let $g : \mathbf{R}^D \to \mathbf{R}$ be λ-strongly convex, and $f_i^* : \mathbf{R}^{N_i} \to \mathbf{R} \cup \{\infty\}$ be γ-strongly convex over its domain. Given any $\tilde{w} \in \mathbf{R}^d$ and $\tilde{\alpha} \in \mathbf{R}^N$, we define the following two functions:*

$$L(w, \alpha) = g(w) + \frac{1}{m}\alpha^T X w - \frac{1}{m}\sum_{i=1}^m f_i * (\alpha_i), \tag{B.4.1}$$

$$L_\delta(w, \alpha) = L(w, \alpha) + \frac{\delta\lambda}{2}\|w - \tilde{w}\|^2 - \frac{\delta\gamma}{2m}\|\alpha - \tilde{\alpha}\|^2. \tag{B.4.2}$$

Let $(w^\star, \alpha^\star)$ and $(\tilde{w}^\star, \tilde{\alpha}^\star)$ be the (unique) saddle points of $L(w, \alpha)$ and $L_\delta(w, \alpha)$, respectively. Then we have

$$\lambda\|\tilde{w} - \tilde{w}^\star\|^2 + \frac{\gamma}{m}\|\tilde{\alpha} - \tilde{\alpha}^\star\|^2 \leq \lambda\|\tilde{w} - w^\star\|^2 + \frac{\gamma}{m}\|\tilde{\alpha} - \alpha^\star\|^2, \tag{B.4.3}$$

$$\left(\lambda\|\tilde{w}^\star - w^\star\|^2 + \frac{\gamma}{m}\|\tilde{\alpha}^\star - \alpha^\star\|^2\right)^{1/2} \leq \frac{\delta}{1+\delta}\left(\lambda\|\tilde{w} - w^\star\|^2 + \frac{\gamma}{m}\|\tilde{\alpha} - \alpha^\star\|^2\right)^{1/2} \tag{B.4.4}$$

Proof: This lemma can be proved using the theory of monotone operators [e.g., 156, 158], as done by Balamurugan and Bach [12]. Here we give an elementary proof based on first-order optimality conditions.

By assumption, we have

$$(w^\star, \alpha^\star) = \arg\min_w \max_\alpha L(w, \alpha),$$
$$(\tilde{w}^\star, \tilde{\alpha}^\star) = \arg\min_w \max_\alpha L_\delta(w, \alpha).$$

Optimality conditions for $(\tilde{w}^\star, \tilde{\alpha}^\star)$ as a saddle point of L_δ:

$$-\frac{1}{m}X^T\tilde{\alpha}^\star - \delta\lambda\left(\tilde{w}^\star - \tilde{w}\right) \in \partial g(\tilde{w}^\star), \tag{B.4.5}$$

$$X\tilde{w}^\star - \delta\gamma\left(\tilde{\alpha}^\star - \tilde{\alpha}\right) \in \partial\sum_{i=1}^m f_i^*(\tilde{\alpha}^\star). \tag{B.4.6}$$

For any $\xi \in \partial g(\tilde{w}^\star)$, it holds that $\xi + \frac{1}{m} X^T \alpha^\star \in \partial_w L(\tilde{w}^\star, \alpha^\star)$. Therefore using (B.4.5) we have

$$\frac{1}{m} X^T(\alpha^\star - \tilde{\alpha}^\star) - \delta\lambda\,(\tilde{w}^\star - \tilde{w}) \;\in\; \partial_w L(\tilde{w}^\star, \alpha^\star).$$

Since $L(w, \alpha^\star)$ is strongly convex in w with convexity parameter λ, we have

$$L(\tilde{w}^\star, \alpha^\star) + \left(\frac{1}{m} X^T(\alpha^\star - \tilde{\alpha}^\star) - \delta\lambda\,(\tilde{w}^\star - \tilde{w})\right)^T (w^\star - \tilde{w}^\star) + \frac{\lambda}{2}\|\tilde{w}^\star - w^\star\|^2 \le L(w^\star, \alpha^\star). \tag{B.4.7}$$

Similarly, we have

$$\frac{1}{m} X^T(\tilde{w}^\star - w^\star) - \frac{\delta\gamma}{m}\,(\tilde{\alpha}^\star - \tilde{\alpha}) \;\in\; \partial_\alpha\left(-L(w^\star, \tilde{\alpha}^\star)\right),$$

and since $-L(w^\star, \alpha)$ is strongly convex in α with convexity parameter $\frac{\gamma}{m}$, we have

$$-L(w^\star, \tilde{\alpha}^\star) + \left(\frac{1}{m} X^T(\tilde{w}^\star - w^\star) - \frac{\delta\gamma}{m}\,(\tilde{\alpha}^\star - \tilde{\alpha})\right)^T (\alpha^\star - \tilde{\alpha}^\star) + \frac{\gamma}{2m}\|\tilde{\alpha}^\star - \alpha^\star\|^2 \le -L(w^\star, \alpha^\star). \tag{B.4.8}$$

Adding inequalities (B.4.7) and (B.4.8) together gives

$$\begin{aligned} &L(\tilde{w}^\star, \alpha^\star) - L(w^\star, \tilde{\alpha}^\star) \\ &+ \delta\lambda(\tilde{w}^\star - \tilde{w})^T(\tilde{w}^\star - w^\star) + \frac{\delta\gamma}{m}(\tilde{\alpha}^\star - \tilde{\alpha})^T(\tilde{\alpha}^\star - \alpha^\star) + \frac{\lambda}{2}\|\tilde{w}^\star - w^\star\|^2 + \frac{\gamma}{2m}\|\tilde{\alpha}^\star - \alpha^\star\|^2 \le 0. \end{aligned}$$

Combining with the inequality

$$L(\tilde{w}^\star, \alpha^\star) - L(w^\star, \tilde{\alpha}^\star) \ge \frac{\lambda}{2}\|\tilde{w}^\star - w^\star\|^2 + \frac{\gamma}{2m}\|\tilde{\alpha}^\star - \alpha^\star\|^2,$$

we obtain

$$\lambda\|\tilde{w}^\star - w^\star\|^2 + \frac{\gamma}{m}\|\tilde{\alpha}^\star - \alpha^\star\|^2 + \delta\lambda(\tilde{w}^\star - \tilde{w})^T(\tilde{w}^\star - w^\star) + \frac{\delta\gamma}{m}(\tilde{\alpha}^\star - \tilde{\alpha})^T(\tilde{\alpha}^\star - \alpha^\star) \le 0. \tag{B.4.9}$$

Proof of the first claim. We can drop the nonnegative terms on the left-hand side of (B.4.9) to obtain

$$\lambda(\tilde{w}^\star - \tilde{w})^T(\tilde{w}^\star - w^\star) + \frac{\gamma}{m}(\tilde{\alpha}^\star - \tilde{\alpha})^T(\tilde{\alpha}^\star - \alpha^\star) \le 0.$$

The two inner product terms on the left-hand side of the inequality above can be expanded as follows:

$$\begin{aligned} (\tilde{w}^\star - \tilde{w})^T(\tilde{w}^\star - w^\star) &= (\tilde{w}^\star - \tilde{w})^T(\tilde{w}^\star - \tilde{w} + \tilde{w} - w^\star) = \|\tilde{w}^\star - \tilde{w}\|^2 + (\tilde{w}^\star - \tilde{w})^T(\tilde{w} - w^\star), \\ (\tilde{\alpha}^\star - \tilde{\alpha})^T(\tilde{\alpha}^\star - \alpha^\star) &= (\tilde{\alpha}^\star - \tilde{\alpha})^T(\tilde{\alpha}^\star - \tilde{\alpha} + \tilde{\alpha} - \alpha^\star) = \|\tilde{\alpha}^\star - \tilde{\alpha}\|^2 + (\tilde{\alpha}^\star - \tilde{\alpha})^T(\tilde{\alpha} - \alpha^\star). \end{aligned}$$

Combining them with the last inequality, we have

$$\begin{aligned} \lambda\|\tilde{w}^\star - \tilde{w}\|^2 + \frac{\gamma}{m}\|\tilde{\alpha}^\star - \tilde{\alpha}\|^2 &\le -\lambda(\tilde{w}^\star - \tilde{w})^T(\tilde{w} - w^\star) - \frac{\gamma}{m}(\tilde{\alpha}^\star - \tilde{\alpha})^T(\tilde{\alpha} - \alpha^\star) \\ &\le \frac{\lambda}{2}\left(\|\tilde{w}^\star - \tilde{w}\|^2 + \|\tilde{w} - w^\star\|^2\right) + \frac{\gamma}{2m}\left(\|\tilde{\alpha}^\star - \tilde{\alpha}\|^2 + \|\tilde{\alpha} - \alpha^\star\|^2\right), \end{aligned}$$

which implies

$$\frac{\lambda}{2}\|\tilde{w}^\star - \tilde{w}\|^2 + \frac{\gamma}{2m}\|\tilde{\alpha}^\star - \tilde{\alpha}\|^2 \;\le\; \frac{\lambda}{2}\|\tilde{w} - w^\star\|^2 + \frac{\gamma}{2m}\|\tilde{\alpha} - \alpha^\star\|^2.$$

Proof of the second claim. We expand the two inner product terms in (B.4.9) as follows:

$$(\tilde{w}^\star - \tilde{w})^T(\tilde{w}^\star - w^\star) = (\tilde{w}^\star - w^\star + w^\star - \tilde{w})^T(\tilde{w}^\star - w^\star) = \|\tilde{w}^\star - w^\star\|^2 + (w^\star - \tilde{w})^T(\tilde{w}^\star - w^\star),$$
$$(\tilde{\alpha}^\star - \tilde{\alpha})^T(\tilde{\alpha}^\star - \alpha^\star) = (\tilde{\alpha}^\star - \alpha^\star + \alpha^\star - \tilde{\alpha})^T(\tilde{\alpha}^\star - \alpha^\star) = \|\tilde{\alpha}^\star - \alpha^\star\|^2 + (\alpha^\star - \tilde{\alpha})^T(\tilde{\alpha}^\star - \alpha^\star).$$

Then (B.4.9) becomes

$$\begin{aligned}
& (1+\delta)\lambda\|\tilde{w}^\star - w^\star\|^2 + (1+\delta)\frac{\Gamma}{m}\|\tilde{w}^\star - w^\star\|^2 \\
\le\;& \delta\lambda(\tilde{w} - w^\star)^T(\tilde{w}^\star - w^\star) + \frac{\delta\gamma}{m}(\tilde{\alpha} - \alpha^\star)^T(\tilde{\alpha}^\star - \alpha^\star) \\
\le\;& \delta\left(\lambda\|\tilde{w} - w^\star\|^2 + \frac{\gamma}{m}\|\tilde{\alpha} - \alpha^\star\|^2\right)^{1/2}\left(\lambda\|\tilde{w}^\star - w^\star\|^2 + \frac{\gamma}{m}\|\tilde{\alpha}^\star - \alpha^\star\|^2\right)^{1/2},
\end{aligned}$$

where in the second inequality we used the Cauchy-Schwarz inequality. Therefore we have

$$\left(\lambda\|\tilde{w}^\star - w^\star\|^2 + \frac{\gamma}{m}\|\tilde{\alpha}^\star - \alpha^\star\|^2\right)^{1/2} \;\le\; \frac{\delta}{1+\delta}\left(\lambda\|\tilde{w} - w^\star\|^2 + \frac{\gamma}{m}\|\tilde{\alpha} - \alpha^\star\|^2\right)^{1/2},$$

which is the desired result. ■

To simplify notations in the rest of the proof, we let $z = (w, \alpha)$ and define

$$\|z\| = \left(\lambda\|w\|^2 + \frac{\gamma}{m}\|\alpha\|^2\right)^{1/2}.$$

The results of Lemma 8 can be written as

$$\|\tilde{z} - \tilde{z}^\star\| \;\le\; \|\tilde{z} - z^\star\|, \tag{B.4.10}$$
$$\|\tilde{z}^\star - z^\star\| \;\le\; \frac{\delta}{1+\delta}\|\tilde{z} - z^\star\|. \tag{B.4.11}$$

Next consider the convergence of Algorithm 9, and follow the proof ideas in Balamurugan and Bach [12, Section D.3].

If we use DSCOVR-SVRG (option 1) in each round of Algorithm 9, then Algorithm 7 is called with initial point $\tilde{z}^{(r)} = (\tilde{w}^{(r)}, \tilde{\alpha}^{(r)})$ and after S stages, it outputs $\tilde{z}^{(r+1)}$ as an approximate saddle point of $L_\delta^{(r)}(w, \alpha)$, which is defined in (6.5.1). Then Theorem 6.3.1 implies

$$\mathbf{E}\big[\|\tilde{z}^{(r+1)} - \tilde{z}^{\star(r)}\|^2\big] \le \left(\frac{2}{3}\right)^S \mathbf{E}\big[\|\tilde{z}^{(r)} - \tilde{z}^{\star(r)}\|^2\big], \tag{B.4.12}$$

where $\tilde{z}^{\star(r)}$ denotes the unique saddle point of $L_\delta^{(r)}(w, \alpha)$. By Minkowski's inequality, we have

$$\big(\mathbf{E}\big[\|\tilde{z}^{(r+1)} - z^\star\|^2\big]\big)^{1/2} \le \big(\mathbf{E}\big[\|\tilde{z}^{(r+1)} - \tilde{z}^{\star(r)}\|^2\big]\big)^{1/2} + \big(\mathbf{E}\big[\|\tilde{z}^{\star(r)} - z^\star\|^2\big]\big)^{1/2},$$

where $z^\star$ is the unique saddle point of $L(w,\alpha)$. Using (B.4.12), (B.4.10) and (B.4.11), we obtain

$$\begin{aligned}\left(\mathbf{E}\left[\|\tilde{z}^{(r+1)}-z^\star\|^2\right]\right)^{1/2} &\leq \left(\frac{2}{3}\right)^{S/2}\left(\mathbf{E}\left[\|\tilde{z}^{(r)}-\tilde{z}^{\star(r)}\|^2\right]\right)^{1/2}+\left(\mathbf{E}\left[\|\tilde{z}^{\star(r)}-z^\star\|^2\right]\right)^{1/2}\\ &\leq \left(\frac{2}{3}\right)^{S/2}\left(\mathbf{E}\left[\|\tilde{z}^{(r)}-z^\star\|^2\right]\right)^{1/2}+\frac{\delta}{1+\delta}\left(\mathbf{E}\left[\|\tilde{z}^{(r)}-z^\star\|^2\right]\right)^{1/2}\\ &= \left[\left(\frac{2}{3}\right)^{S/2}+\frac{\delta}{1+\delta}\right]\left(\mathbf{E}\left[\|\tilde{z}^{(r)}-z^\star\|^2\right]\right)^{1/2}, \qquad \text{(B.4.13)}\end{aligned}$$

Therefore, if $S \geq \frac{2\log(2(1+\delta))}{\log(3/2)}$, we have

$$\left(\frac{2}{3}\right)^{S/2}+\frac{\delta}{1+\delta} \leq \frac{1}{2(1+\delta)}+\frac{\delta}{1+\delta}=\frac{1+2\delta}{2(1+\delta)}=1-\frac{1}{2(1+\delta)},$$

which implies

$$\mathbf{E}\left[\|\tilde{z}^{(r+1)}-z^\star\|^2\right] \leq \left(1-\frac{1}{2(1+\delta)}\right)^2\mathbf{E}\left[\|\tilde{z}^{(r)}-z^\star\|^2\right]. \qquad \text{(B.4.14)}$$

If we use DISCOVR-SAGA (option 2) in Algorithm 9, then Algorithm 8 is called with initial point $\tilde{z}^{(r)}=(\tilde{w}^{(r)},\tilde{\alpha}^{(r)})$ and after M steps, it outputs $\tilde{z}^{(r+1)}$ as an approximate saddle point of $L_\delta^{(r)}(w,\alpha)$. Then Theorem 6.4.1 implies

$$\mathbf{E}\left[\|\tilde{z}^{(r+1)}-\tilde{z}^{\star(r)}\|^2\right] \leq \frac{4}{3}\left(1-\frac{1}{3\Gamma_\delta}\right)^M\mathbf{E}\left[\|\tilde{z}^{(r)}-\tilde{z}^{\star(r)}\|^2\right].$$

Using similar arguments as in (B.4.13), we have

$$\left(\mathbf{E}\left[\|\tilde{z}^{(r+1)}-z^\star\|\right]^2\right)^{1/2} \leq \left[\frac{4}{3}\left(1-\frac{1}{3\Gamma_\delta}\right)^{M/2}+\frac{\delta}{1+\delta}\right]\left(\mathbf{E}\left[\|\tilde{z}^{(r)}-z^\star\|^2\right]\right)^{1/2}.$$

Therefore, if $M \geq 6\log\left(\frac{8(1+\delta)}{3}\right)\Gamma_\delta$, we have

$$\frac{4}{3}\left(1-\frac{1}{3\Gamma_\delta}\right)^{M/2}+\frac{\delta}{1+\delta} \leq \frac{1}{2(1+\delta)}+\frac{\delta}{1+\delta}=\frac{1+2\delta}{2(1+\delta)}=1-\frac{1}{2(1+\delta)},$$

which implies the same inequality in (B.4.14).

In summary, using either option 1 or option 2 in Algorithm 9, we have

$$\mathbf{E}\left[\|\tilde{z}^{(r)}-z^\star\|^2\right] \leq \left(1-\frac{1}{2(1+\delta)}\right)^{2r}\|\tilde{z}^{(0)}-z^\star\|^2.$$

In order to have $\mathbf{E}\left[\|\tilde{z}^{(r)}-z^\star\|^2\right] \leq \epsilon$, it suffices to have $r \geq (1+\delta)\log\left(\|\tilde{z}^{(0)}-z^\star\|^2/\epsilon\right)$.

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Gregory S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian J. Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Józefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Gordon Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul A. Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda B. Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *CoRR*, abs/1603.04467, 2016. URL `http://arxiv.org/abs/1603.04467`. 3.5.1

[2] Alekh Agarwal and John C Duchi. Distributed delayed stochastic optimization. In *Advances in Neural Information Processing Systems (NIPS) 24*, pages 873–881, 2011. 6.2.2

[3] Zeyuan Allen-Zhu. Katyusha: Accelerated variance reduction for faster sgd. *ArXiv e-prints, abs/1603.05953*, 2016. 5.2

[4] Zeyuan Allen-Zhu. Katyusha: The first direct acceleration of stochastic gradient methods. In *Proceedings of 49th Annual ACM Symposium on the Theory of Computing (STOC)*, pages 1200–1205, 2017. 6.2.1

[5] Zeyuan Allen-Zhu, Peter Richtárik, Zheng Qu, and Yang Yuan. Even faster accelerated coordinate descent using non-uniform sampling. *arXiv preprint arXiv:1512.09103*, 2016. 5.2, 5.3

[6] Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, and Michael Collins. Globally normalized transition-based neural networks. *arXiv preprint arXiv:1603.06042*, 2016. 2.1

[7] Yossi Arjevani and Ohad Shamir. Communication complexity of distributed convex learning and optimization. In *Advances in Neural Information Processing Systems (NIPS) 28*, pages 1756–1764. 2015. 6.2.2, 6.9

[8] Arda Aytekin, Hamid Reza Feyzmahdavian, and Mikael Johansson. Analysis and implementatino of an asynchronous optimization algorithm for the parameter server. arXiv:1610.05507, 2016. 6.2.2

[9] Jimmy Ba, Volodymyr Mnih, and Koray Kavukcuoglu. Multiple object recognition with visual attention. *arXiv preprint arXiv:1412.7755*, 2014. 2.2

[10] Lei Jimmy Ba, Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *CoRR*,

abs/1607.06450, 2016. URL http://arxiv.org/abs/1607.06450. 3.3.2

[11] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations*, 2015. 2.1, 2.2, 3.1, 3.4

[12] Palaniappan Balamurugan and Francis Bach. Stochastic variance reduction methods for saddle-point problems. In *Advances in Neural Information Processing Systems (NIPS) 29*, pages 1416–1424, 2016. 5.2, 6.2.1, 6.2.2, 6.5, B.4, B.4

[13] Maria Florina Balcan, Simon S. Du, Yining Wang, and Adams Wei Yu. An Improved Gap-Dependency Analysis of the Noisy Power Method. In *COLT*, 2016. 1.4

[14] A. Beck and M. Teboulle. A fast iterative shrinkage-threshold algorithm for linear inverse problems. *SIAM Journal on Imaging Sciences*, 2(1):183–202, 2009. 6.2.1

[15] Yoshua Bengio. Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127, 2009. 4.1

[16] Dimitri P. Bertsekas and John N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods*. Prentice-Hall, 1989. 6.1

[17] Stephen P. Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1):1–122, 2011. 6.2.2, 6.8, 6.8

[18] A Chambolle and T Pock. A first-order primal-dual algorithm for convex problems with applications to imaging. *Journal of Mathematical Imaging and Vision*, 40(1):120–145, 2011. 5.2

[19] Antonin Chambolle and Thomas Pock. A first-order primal-dual algorithm for convex problems with applications to imaging. *Journal of Mathematical Imaging and Vision*, 40 (1):120–145, 2011. 6.2.1

[20] Antonin Chambolle and Thomas Pock. On the ergodic convergence rates of a first-order primal–dual algorithm. *Mathematical Programming*, pages 1–35, 2015. 6.2.1

[21] William Chan, Navdeep Jaitly, Quoc V Le, and Oriol Vinyals. Listen, attend and spell. *arXiv preprint arXiv:1508.01211*, 2015. 2.2

[22] Danqi Chen, Jason Bolton, and Christopher D. Manning. A thorough examination of the cnn/daily mail reading comprehension task. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*, 2016. 2.4.4

[23] Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. Reading wikipedia to answer open-domain questions. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, pages 1870–1879, 2017. (document), 3.2, 3.3.2, 3.5.1, 3.5.1, 3.2

[24] Weizhu Chen, Zhenghao Wang, and Jingren Zhou. Large-scale L-BFGS using MapReduce. In *Advances in Neural Information Processing Systems (NIPS) 27*, pages 1332–1340. 2014. 6.2.2

[25] Eunsol Choi, Daniel Hewlett, Alexandre Lacoste, Illia Polosukhin, Jakob Uszkoreit, and Jonathan Berant. Hierarchical question answering for long documents. *arXiv preprint arXiv:1611.01839*, 2016. 2.2

[26] François Chollet. Xception: Deep learning with depthwise separable convolutions. *CoRR*, abs/1610.02357, 2016. URL `http://arxiv.org/abs/1610.02357`. 3.3.2

[27] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014. 3.2

[28] Junyoung Chung, Sungjin Ahn, and Yoshua Bengio. Hierarchical multiscale recurrent neural networks. *arXiv preprint arXiv:1609.01704*, 2016. 2.2

[29] Christopher Clark and Matt Gardner. Simple and effective multi-paragraph reading comprehension. *CoRR*, abs/1710.10723, 2017. URL `http://arxiv.org/abs/1710.10723`. 3.5.2

[30] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12(Aug):2493–2537, 2011. 2.1

[31] Dominik Csiba and Peter Richtárik. Importance sampling for minibatches. *arXiv preprint arXiv:1602.02283*, 2016. 5.3

[32] Dominik Csiba, Zheng Qu, and Peter Richtarik. Stochastic dual coordinate ascent with adaptive probabilities. In *ICML*, 2015. 5.3

[33] Yiming Cui, Zhipeng Chen, Si Wei, Shijin Wang, Ting Liu, and Guoping Hu. Attention-over-attention neural networks for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, pages 593–602, 2017. (document), 3.2, 3.2

[34] Andrew M. Dai and Quoc V. Le. Semi-supervised sequence learning. In *Advances in Neural Information Processing Systems*, pages 3079–3087, 2015. 2.1

[35] Bo Dai, Bo Xie, Niao He, Yingyu Liang, Anant Raj, Maria-Florina Balcan, and Le Song. Scalable kernel methods via doubly stochastic gradients. In *NIPS*, pages 3041–3049, 2014. 5.2

[36] Cong Dang and Guanghui Lan. Randomized first-order methods for saddle point optimization. Technical report, 2014. 5.2

[37] Cong D. Dang and Guanghui Lan. Stochastic block mirror descent methods for nonsmooth and stochastic optimization. *SIAM Journal on Optimization*, 25(2):856–881, 2015. 5.2

[38] Jeff Dean and Sanjay Ghemawat. MapReduce: Simplfied data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008. 6.1

[39] Aaron Defazio, Francis Bach, and Simon Lacoste-Julien. SAGA: A fast incremental gradient method with support for non-strongly convex composite objectives. In *Advances in Neural Information Processing Systems (NIPS) 27*, pages 1646–1654. 2014. 5.2, 6.2, 6.2.1, 6.2.1, 6.3, 6.4

[40] Aaron Defazio, Justin Domke, and Tibrio S. Caetano. Finito: A faster, permutable incremental gradient method for big data problems. In *ICML*, pages 1125–1133, 2014. 5.2

[41] Qi Deng, Guanghui Lan, and Anand Rangarajan. Randomized block subgradient methods for convex nonsmooth and stochastic optimization. Technical report, 2015. 5.2

[42] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018. 1.1

[43] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In *NAACL*, pages 4171–4186, 2019. 3.5.1

[44] Li Dong, Jonathan Mallinson, Siva Reddy, and Mirella Lapata. Learning to paraphrase for question answering. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 875–886. Association for Computational Linguistics, 2017. URL `http://aclweb.org/anthology/D17-1091`. 3.4

[45] Petros Drineas, Michael W. Mahoney, S. Muthukrishnan, and Tamàs Sarlós. Faster least squares approximation. *Numerische Mathematik*, 117(2):219–249, feb 2011. 5.4.2

[46] John Duchi, Michael I Jordan, and Brendan McMahan. Estimation, optimization, and parallelism when data is sparse. In *NIPS*, pages 2832–2840, 2013. 4.5

[47] John C. Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, 2011. 4.1, 4.3, 4.4, 4.4, 4.4

[48] John C Duchi, Alekh Agarwal, and Martin J Wainwright. Dual averaging for distributed optimization: convergence analysis and network scaling. *IEEE Transactions on Automatic Control*, 57(3):592–606, 2012. 4.2, 6.1

[49] R.-E. Fan and C.-J. Lin. LIBSVM data: Classification, regression and multi-label. URL: http://www.csie.ntu.edu.tw/˜cjlin/libsvmtools/datasets, 2011. 6.8

[50] Olivier Fercoq and Peter Richtrik. Accelerated, parallel and proximal coordinate descent. *CoRR*, abs/1312.5799, 2013. 5.2

[51] Roy Frostig, Rong Ge, Sham Kakade, and Aaron Sidford. Un-regularizing: approximate proximal point and faster stochastic algorithms for empirical risk minimization. In *Proceedings of The 32nd International Conference on Machine Learning (ICML)*, pages 2540–2548. 2015. 6.5

[52] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. Convolutional sequence to sequence learning. In *International Conference on Machine Learning*, 2017. 3.2

[53] Yichen Gong and Samuel R. Bowman. Ruminating reader: Reasoning with gated multi-hop attention. *CoRR*, abs/1704.07415, 2017. URL `http://arxiv.org/abs/1704.07415`. (document), 3.2, 3.7

[54] Alex Graves. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016. 2.2

[55] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014. 2.2

[56] Michael Hahn and Frank Keller. Modeling human reading with neural attention. In *EMNLP*, pages 85–95, 2016. 2.2

[57] N. Halko, P. G. Martinsson, and J. A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Review*, 53(2):217–288, May 2011. 5.4.2

[58] Robert Hannah and Wotao Yin. More iterations per second, same quality — why asynchronous algorithms may drastically outperform traditional ones. CAM Report 17-50, University of California at Los Angeles, 2017. 6.2.2

[59] Elad Hazan, Kfir Y. Levy, and Shai Shalev-Shwartz. Beyond convexity: Stochastic quasi-convex optimization. In *NIPS*, pages 1594–1602, 2015. 4.1, 4.2

[60] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV*, 2015. (document), 4.1, 4.5.2, 4.2

[61] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 1.1, 4.5.2, 4.5.2

[62] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. *CoRR*, abs/1603.05027, 2016. 4.5.3

[63] Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. In *Advances in Neural Information Processing Systems*, pages 1693–1701, 2015. 2.1, 3.2

[64] Daniel Hewlett, Alexandre Lacoste, Llion Jones, Illia Polosukhin, Andrew Fandrianto, Jay Han, Matthew Kelcey, and David Berthelot. Wikireading: A novel large-scale language understanding task over wikipedia. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*, 2016. 3.2

[65] Felix Hill, Antoine Bordes, Sumit Chopra, and Jason Weston. The goldilocks principle: Reading children's books with explicit memory representations. *CoRR*, abs/1511.02301, 2015. 2.1, 2.4.4, 3.2

[66] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Lecture 6a: Overview of mini-batch gradient descent. *Neural Networks for Machine Learning*. 4.2, 4.3

[67] Geoffrey Hinton, George Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Brian Kingsbury, and Tara Sainath. Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal Processing Magazine*, 29:82–97, November 2012. 1.1

[68] J.-B. Hiriart-Urruty and C. Lemaréchal. *Fundamentals of Convex Analysis*. Springer, 2001. 6.2.1

[69] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997. 2.1, 2.2, 2.4, 3.2, 4.5.4

[70] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, and Jürgen Schmidhuber. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. In S. C. Kremer and J. F. Kolen, editors, *A Field Guide to Dynamical Recurrent Neural Networks*. IEEE press, 2001. 2.2

[71] Minghao Hu, Yuxing Peng, and Xipeng Qiu. Reinforced mnemonic reader for machine comprehension. *CoRR*, abs/1705.02798, 2017. URL `http://arxiv.org/abs/1705.02798`. (document), 3.1, 3.2, 3.5.2, 3.2, 3.7, 3.8

[72] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q. Weinberger. Deep networks with stochastic depth. In *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part IV*, pages 646–661, 2016. 3.3.2, 3.5.1

[73] Lei Huang, Xianglong Liu, Bo Lang, Adams Wei Yu, and Bo Li. Orthogonal weight normalization: Solution to optimization over multiple dependent stiefel manifolds in deep neural networks. In *AAAI*, 2018. 1.4

[74] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, pages 448–456, 2015. 4.2, 4.5.2

[75] Martin Jaggi, Virginia Smith, Martin Takac, Jonathan Terhorst, Sanjay Krishnan, Thomas Hofmann, and Michael I Jordan. Communication-efficient distributed dual coordinate ascent. In *Advances in Neural Information Processing Systems (NIPS) 27*, pages 3068–3076. 2014. 6.2.2

[76] Yacine Jernite, Edouard Grave, Armand Joulin, and Tomas Mikolov. Variable computation in recurrent neural networks. *arXiv preprint arXiv:1611.06188*, 2016. 2.2

[77] Robin Jia and Percy Liang. Adversarial examples for evaluating reading comprehension systems. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017*, pages 2021–2031, 2017. 3.5.1, 14

[78] R. Johnson and T. Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in Neural Information Processing Systems (NIPS) 26*, pages 315–323. 2013. 5.2, 6.2, 6.2.1, 6.2.1, 6.3, 6.3

[79] Mandar Joshi, Eunsol Choi, Daniel S. Weld, and Luke Zettlemoyer. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, pages 1601–1611, 2017. (document), 3.2, 3.5, 3.5.2, 3.8

[80] Lukasz Kaiser, Aidan N Gomez, and Francois Chollet. Depthwise separable convolutions for neural machine translation. *arXiv preprint arXiv:1706.03059*, 2017. 3.3.2

[81] Nal Kalchbrenner and Phil Blunsom. Recurrent continuous translation models. In *EMNLP*, 2013. 2.1

[82] Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014. 2.1, 3.2, 4.5.5

[83] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL `http://arxiv.org/abs/1412.6980`. 2.4, 3.5.1, 4.1, 4.2, 4.3, 4.5

[84] Krzysztof C Kiwiel. Convergence and efficiency of subgradient methods for quasiconvex minimization. *Mathematical programming*, 90(1):1–25, 2001. 4.2

[85] Jakub Konecný and Peter Richtárik. Semi-stochastic gradient descent methods. *CoRR*, abs/1312.1666, 2013. 5.2

[86] Jakub Konecný, Zheng Qu, and Peter Richtárik. Semi-stochastic coordinate descent. *CoRR*, abs/1412.6293, 2014. 5.2

[87] Jan Koutnik, Klaus Greff, Faustino Gomez, and Juergen Schmidhuber. A clockwork rnn. In *International Conference on Machine Learning*, 2014. 2.2

[88] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, pages 1106–1114, 2012. 1.1

[89] Guanghui Lan and Yi Zhou. An optimal randomized incremental gradient method. Technical report, Department of Industrial and System Engineering, University of Florida, July 2015. 5.2, 6.2.1, 6.6, 6.6

[90] Quoc V. Le and Tomas Mikolov. Distributed representations of sentences and documents. In *International Conference on Machine Learning (ICML)*, 2014. 2.1

[91] N. Le Roux, M. Schmidt, and F. Bach. A stochastic gradient method with an exponential convergence rate for finite training sets. In *Advances in Neural Information Processing Systems 25*, pages 2672–2680. 2012. 5.2, 6.2.1, 6.3

[92] Yann Lecun, Leon Bottou, Yoshua Bengio, and Patrick Haffner? Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324, 1998. 4.5.1

[93] Ching-Pei Lee, Po-Wei Wang, Weizhu Chen, and Chih-Jen Lin. Limited-memory common-directions method for distributed optimization and its application on empirical risk minimization. In *Proceedings of the 2017 SIAM International Conference on Data Mining*, pages 732–740, 2017. 6.2.2

[94] Jason D. Lee, Qihang Lin, Tengyu Ma, and Tianbao Yang. Distributed stochastic variance reduced gradient methods and a lower bound for communication complexity. arXiv:1507.07595, 2015. 6.2.2

[95] Kenton Lee, Tom Kwiatkowski, Ankur P. Parikh, and Dipanjan Das. Learning recurrent span representations for extractive question answering. *CoRR*, abs/1611.01436, 2016. (document), 2.1, 3.2, 3.7

[96] Tao Lei, Regina Barzilay, and Tommi Jaakkola. Rationalizing neural predictions. *arXiv preprint arXiv:1606.04155*, 2016. 2.2

[97] Kfir Y. Levy. The power of normalization: Faster evasion of saddle points. *CoRR*, abs/1611.04831, 2016. URL `http://arxiv.org/abs/1611.04831`. 4.1, 4.2

[98] Mu Li, David G. Andersen, Jun Woo Park, Alexander J. Smola, Amr Ahmed, Vanja Josi-

fovski, James Long, Eugene J. Shekita, and Bor-Yiing Su. Scaling distributed machine learning with the parameter server. In *Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 583–598, 2014. 6.1, 6.2.2

[99] Chen Liang, Jonathan Berant, Quoc Le, Kenneth D. Forbus, and Ni Lao. Neural symbolic machines: Learning semantic parsers on freebase with weak supervision. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017: Long Papers*, 2017. 2.2

[100] C.-Y. Lin, C.-H. Tsai, C.-P. Lee, and C.-J. Lin. Large-scale logistic regression and linear support vector machines using Spark. In *Proceedings of the IEEE Conference on Big Data*, Washington DC, USA, 2014. 6.1, 6.2.2

[101] Hongzhou Lin, Julien Mairal, and Zaid Harchaoui. A universal catalyst for first-order optimization. In *Advances in Neural Information Processing Systems (NIPS) 28*, pages 3384–3392. 2015. 6.5

[102] Qihang Lin and Lin Xiao. An adaptive accelerated proximal gradient method and its homotopy continuation for sparse optimization. *Computatoinal Optimization and Applications*, 60(3):633–674, 2015. 6.8

[103] Qihang Lin, Zhaosong Lu, and Lin Xiao. An accelerated randomized proximal coordinate gradient method and its application to regularized empirical risk minimization. *SIAM Journal on Optimization*, 2015. 5.2, 5.3, 5.4.1

[104] J. Liu, S. J. Wright, C. Ré, V. Bittorf, and S. Sridhar. An asynchronous parallel stochastic coordinate descent algorithm. In *Proceedings of the 31st International Conference on Machine Learning (ICML)*, pages 469–477, 2014. 6.2.2

[105] Rui Liu, Junjie Hu, Wei Wei, Zi Yang, and Eric Nyberg. Structural embedding of syntactic trees for machine comprehension. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017*, pages 826–835, 2017. (document), 3.2, 3.7

[106] Zhaosong Lu and Lin Xiao. On the complexity analysis of randomized block-coordinate descent methods. *Mathematical Programming*, 152:615–642, 2015. 5.2, 5.3, 5.4.1

[107] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. In *EMNLP*, 2015. 3.4

[108] Minh-Thang Luong, Eugene Brevdo, and Rui Zhao. Neural machine translation (seq2seq) tutorial. *https://github.com/tensorflow/nmt*, 2017. 3.4

[109] Chenxin Ma, Virginia Smith, Martin Jaggi, Michael I. Jordan, Peter Richtárik, and Martin Takáč. Adding vs. averaging in distributed primal-dual optimization. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning (ICML)*, pages 1973–1982, 2015. 6.2.2, 6.8

[110] Chenxin Ma, Virginia Smith, Martin Jaggi, Michael I. Jordan, Peter Richtárik, and Martin Takáč. Distributed optimization with arbitrary local solvers. *Optimization Methods and Software*, 32(4):813–848, 2017. 6.8

[111] Andrew L Maas, Raymond E Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and

Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 142–150. Association for Computational Linguistics, 2011. 2.1, 2.4.2

[112] Michael W. Mahoney. Randomized algorithms for matrices and data. *Foundations and Trends in Machine Learning*, 3(2):123–224, 2011. 5.4.2

[113] Julien Mairal. Incremental majorization-minimization optimization with application to large-scale machine learning. *SIAM Journal on Optimization*, 25:829–855, 2015. 5.2

[114] Jonathan Mallinson, Rico Sennrich, and Mirella Lapata. Paraphrasing revisited with neural machine translation. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 881–893. Association for Computational Linguistics, 2017. URL `http://aclweb.org/anthology/E17-1083`. 3.4

[115] Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of english: The penn treebank. *Computational Linguistics*, 19(2):313–330, 1993. 4.5.4

[116] Shin Matsushima, Hyokun Yun, Xinhua Zhang, and S. V. N. Vishwanathan. Distributed stochastic optimization of the regularized risk. arXiv:1406.4363, 2014. 5.2, 6.2

[117] Bryan McCann, James Bradbury, Caiming Xiong, and Richard Socher. Learned in translation: Contextualized word vectors. In *NIPS*, pages 6294–6305, 2017. 3.5.1

[118] H. Brendan McMahan and Matthew J. Streeter. Delay-tolerant algorithms for asynchronous distributed online learning. In *Advances in Neural Information Processing Systems (NIPS) 27*, pages 2915–2923, 2014. 6.2.2

[119] Xiangrui Meng, Joseph Bradley, Burak Yavuz, Evan Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, DB Tsai, Manish Amde, Sean Owen, Doris Xin, Reynold Xin, Michael J. Franklin, Reza Zadeh, Matei Zaharia, and Ameet Talwalkar. MLlib: Machine learning in Apache Spark. *Journal of Machine Learning Research*, 17(34):1–7, 2016. 6.1

[120] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013. 2.4.2

[121] Volodymyr Mnih, Nicolas Heess, Alex Graves, et al. Recurrent models of visual attention. In *Advances in neural information processing systems*, pages 2204–2212, 2014. 2.2, 2.3.2

[122] MPI Forum. MPI: a message-passing interface standard, Version 3.0. Document available at `http://www.mpi-forum.org`, 2012. 6.1, 6.7

[123] Ramesh Nallapati, Bowen Zhou, Caglar Gulcehre, Bing Xiang, et al. Abstractive text summarization using sequence-to-sequence RNNs and beyond. In *Conference on Computational Natural Language Learning (CoNLL)*, 2016. 2.1

[124] Angelia Nedić and Asuman Ozdaglar. Distributed subgradient methods for multi-agent optimization. *IEEE Transactions on Automatic Control*, 54(1):48–61, January 2009. 6.1

[125] Angelia Nedić, Alex Olshevsky, and Wei Shi. Achieving geometric convergence for distributed optimizaiton over time-varying graphs. arXiv:1607.03218, 2016. 6.1

[126] Arkadi Nemirovski. Prox-method with rate of convergence $o(1/t)$ for variational inequalities with lipschitz continuous monotone operators and smooth convex-concave saddle-point problems. *SIAM Journal on Optimization*, 15(1):229–251, 2004. 5.2

[127] Nesterov. Minimization methods for nonsmooth convex and quasiconvex functions. *Matekon*, 29:519–531, 1984. 4.2

[128] Y. Nesterov. *Introductory Lectures on Convex Optimization: A Basic Course*. Kluwer, Boston, 2004. 5.2, 5.4.1, 6.2.1, 6.8

[129] Y. Nesterov. Efficiency of coordinate descent methods on huge-scale optimization problems. *SIAM Journal on Optimization*, 22(2):341–362, 2012. 5.2, 6.8

[130] Yu. Nesterov. Smooth minimization of nonsmooth functions. *Mathematical Programming*, 103:127–152, 2005. 5.2, 5.4.1, A.3, A.3

[131] Yu. Nesterov. Gradient methods for minimizing composite functions. *Mathematical Programming, Ser. B*, 140:125–161, 2013. 6.2.1, 6.8

[132] Yurii Nesterov and Sebastian Stich. Efficiency of accelerated coordinate descent method on structured optimization problems. Technical report, Université catholique de Louvain, Center for Operations Research and Econometrics (CORE), 2016. 5.2

[133] Behnam Neyshabur, Ruslan Salakhutdinov, and Nathan Srebro. Path-sgd: Path-normalized optimization in deep neural networks. In *NIPS*, pages 2422–2430, 2015. 4.2

[134] Atsushi Nitanda. Stochastic proximal gradient descent with acceleration techniques. In *NIPS*, 2014. 5.2

[135] OpenMP Architecture Review Board. OpenMP Application Program Interface, Version 3.1. Available at `http://www.openmp.org`, July 2011. 6.8

[136] Boyuan Pan, Hao Li, Zhou Zhao, Bin Cao, Deng Cai, and Xiaofei He. MEMEN: multi-layer embedding with memory networks for machine comprehension. *CoRR*, abs/1707.09098, 2017. (document), 3.5.2, 3.8

[137] Bo Pang and Lillian Lee. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. In *Proceedings of the 43rd annual meeting on association for computational linguistics*, pages 115–124. Association for Computational Linguistics, 2005. 2.1, 4.5.5

[138] Shibin Parameswaran and Kilian Q. Weinberger. Large margin multi-task metric learning. In *NIPS*, 2010. 5.3, 5.5.2, 5.5.3, 5.5.3, 5.6.3

[139] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *ICML*, pages 1310–1318, 2013. 4.2

[140] Zhimin Peng, Yangyang Xu, Ming Yan, and Wotao Yin. ARock: An algorithmic framework for asynchronous parallel coordinate updates. *SIAM Journal on Scientific Computing*, 38(5):2851–2879, 2016. 6.2.2

[141] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vec-

tors for word//w representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014. URL http://www.aclweb.org/anthology/D14-1162. 3.3.2, 3.5.1

[142] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *NAACL*, pages 2227–2237, 2018. 3.5.1

[143] Vu Pham and Laurent El Ghaoui. Robust sketching for multiple square-root lasso problems. In *AISTATS*, 2015. 5.4.2

[144] Zheng Qu and Peter Richtárik. Coordinate descent with arbitrary sampling ii: Expected separable overapproximation. *Optimization Methods Software*, 31(5):858–884, 2016. 5.2, 5.3

[145] Zheng Qu and Peter Richtárik. Coordinate descent with arbitrary sampling i: Algorithms and complexity. *Optimization Methods and Software*, 31(5):829–857, 2016. 5.2, 5.3

[146] Jonathan Raiman and John Miller. Globally normalized reader. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017*, pages 1070–1080, 2017. 3.2, 3.4

[147] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100, 000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016*, pages 2383–2392, 2016. (document), 3.1, 3.2, 3.5, 3.5.1, 3.2, 3.7

[148] Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. Sequence level training with recurrent neural networks. *CoRR*, abs/1511.06732, 2015. URL http://arxiv.org/abs/1511.06732. 2.2

[149] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in Neural Information Processing Systems (NIPS) 24*, pages 693–701, 2011. 6.2.2

[150] Sashank J. Reddi, Ahmed Hefny, Suvrit Sra, Barnabás Póczós, and Alexander J Smola. On variance reduction in stochastic gradient descent and its asynchronous variants. In *Advances in Neural Information Processing Systems (NIPS) 28*, pages 2647–2655. 2015. 6.2.2

[151] Sashank J. Reddi, Jakub Konečný, Peter Richtárik, Barnabás Póczós, and Alex Smola. AIDE: Fast and communication efficient distributed optimization. arXiv:1608.06879, 2016. 6.2.2

[152] P. Richtárik and M. Takáč. Iteration complexity of randomized block-coordinate descent methods for minimizing a composite function. *Mathematical Programming*, 144(1):1–38, 2014. 5.2, 6.8

[153] P. Richtárik and M. Takáč. Parallel coordinate descent methods for big data optimization. *Mathematical Programming*, 156(1):433–484, 2016. 6.2.2

[154] Peter Richtárik and Martin Takáč. On optimal probabilities in stochastic coordinate descent methods. *Optimization Letters*, 10(6):1233–1243, 2016. 5.2, 5.3

[155] R. T. Rockafellar. *Convex Analysis*. Princeton University Press, 1970. 1

[156] R. Tyrrell Rockafellar. Monotone operators and the proximal point algorithm. *SIAM Journal on Control and Optimization*, 14(5), 1976. 6.5, B.4

[157] Alexander M Rush, Sumit Chopra, and Jason Weston. A neural attention model for abstractive sentence summarization. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2015. 2.1

[158] Ernest K. Ryu and Stephen P. Boyd. A primer on monotone operator methods. *Applied and Computational Mathematics: an International Journal*, 15(1):3–43, 2016. B.4

[159] Tim Salimans and Diederik P. Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *NIPS*, 2016. 4.2

[160] Kevin Scaman, Francis Bach, Sébastien Bubeck, Yin Tat Lee, and Laurent Massoulié. Optimal algorithms for smooth and strongly convex distributed optimization in networks. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, pages 3027–3036, Sydney, Australia, 2017. 6.1, 6.2.2, 6.9

[161] Mark W. Schmidt, Nicolas Le Roux, and Francis R. Bach. Minimizing finite sums with the stochastic average gradient. *CoRR*, abs/1309.2388, 2013. 5.2

[162] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2015. 2.1

[163] Rico Sennrich, Barry Haddow, and Alexandra Birch. Improving neural machine translation models with monolingual data. In *ACL (1)*. The Association for Computer Linguistics, 2016. 3.4

[164] Min Joon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. *CoRR*, abs/1611.01603, 2016. URL `http://arxiv.org/abs/1611.01603`. (document), 2.1, 3.1, 3.2, 3.3.2, 3.3.2, 3.3.2, 3.3.2, 3.5.1, 3.5.1, 3.2, 3.4, 3.7, 3.8

[165] Pierre Sermanet, Andrea Frome, and Esteban Real. Attention for fine-grained categorization. *arXiv preprint arXiv:1412.7054*, 2014. 2.2

[166] S. Shalev-Shwartz and T. Zhang. Stochastic dual coordinate ascent methods for regularized loss minimization. *Journal of Machine Learning Research*, 14:567–599, 2013. 5.2, 5.6, 5.6.2, 6.3, 6.8, 6.9

[167] Shai Shalev-Shwartz and Ambuj Tewari. Stochastic methods for l1 regularized loss minimization. In *ICML*, volume 382, 2009. 5.2

[168] Shai Shalev-Shwartz and Tong Zhang. Accelerated mini-batch stochastic dual coordinate ascent. In *NIPS*, pages 378–385, 2013. 5.2, 5.3, 5.4.1

[169] Ohad Shamir, Nati Srebro, and Tong Zhang. Communication-efficient distributed optimization using an approximate newton-type method. In *Proceedings of the 31st International Conference on Machine Learning (ICML)*, pages 1000–1008, Bejing, China, 2014. 6.2.2

[170] Lifeng Shang, Zhengdong Lu, and Hang Li. Neural responding machine for short-text conversation. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2015. 2.1

[171] Tao Shen, Tianyi Zhou, Guodong Long, Jing Jiang, Shirui Pan, and Chengqi Zhang. Disan: Directional self-attention network for rnn/cnn-free language understanding. *CoRR*, abs/1709.04696, 2017. URL `http://arxiv.org/abs/1709.04696`. 3.2

[172] Yelong Shen, Po-Sen Huang, Jianfeng Gao, and Weizhu Chen. Reasonet: Learning to stop reading in machine comprehension. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Halifax, NS, Canada, August 13 - 17, 2017*, pages 1047–1055, 2017. (document), 2.2, 3.2, 3.2, 3.7

[173] Wei Shi, Qing Ling, Gang Wu, and Wotao Yin. EXTRA: An exact first-order algorithm for decentralized consensus optimization. *SIAM Journal on Optimization*, 25(2):944–966, 2015. 6.2.2

[174] Bharat Singh, Soham De, Yangmuzi Zhang, Thomas Goldstein, and Gavin Taylor. Layer-specific adaptive learning rates for deep networks. *CoRR*, abs/1510.04609, 2015. 4.2

[175] Martin Slawski, Ping Li, and Matthias Hein. Regularization-free estimation in trace regression with symmetric positive semidefinite matrices. In *NIPS*, 2015. 5.3, 5.5.2

[176] Richard Socher, Jeffrey Pennington, Eric H. Huang, Andrew Y. Ng, and Christopher D. Manning. Semi-supervised recursive autoencoders for predicting sentiment distributions. In *Proceedings of the conference on empirical methods in natural language processing*, 2011. 2.1

[177] Richard Socher, Alex Perelygin, Jean Y. Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, Christopher Potts, et al. Recursive deep models for semantic compositionality over a sentiment treebank. In *EMNLP*, 2013. 2.1

[178] Alessandro Sordoni, Michel Galley, Michael Auli, Chris Brockett, Yangfeng Ji, Margaret Mitchell, Jian-Yun Nie, Jianfeng Gao, and Bill Dolan. A neural network approach to context-sensitive generation of conversational responses. *arXiv preprint arXiv:1506.06714*, 2015. 2.1

[179] Suvrit Sra, Adams Wei Yu, Mu Li, and Alexander J. Smola. Adadelay: Delay adaptive distributed stochastic optimization. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, pages 957–965, 2016. 1.4, 6.2.2

[180] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *CoRR*, abs/1505.00387, 2015. URL `http://arxiv.org/abs/1505.00387`. 3.3.2

[181] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014. 2.1

[182] Adam Trischler, Zheng Ye, Xingdi Yuan, Jing He, Phillip Bachman, and Kaheer Suleman. A parallel-hierarchical model for machine comprehension on sparse data. *arXiv preprint arXiv:1603.08884*, 2016. 2.1

[183] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N.

Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017. URL http://arxiv.org/abs/1706.03762. (document), 3.2, 3.3.2, 3.3.2, 3.1

[184] Oriol Vinyals and Quoc Le. A neural conversational model. *arXiv preprint arXiv:1506.05869*, 2015. 2.1

[185] Jialei Wang and Lin Xiao. Exploiting strong convexity from data with primal-dual first-order algorithms. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, pages 3694–3702, Sydney, Australia, 2017. 6.6, 6.6, 6.9

[186] Jialei Wang, Jason D Lee, Mehrdad Mahdavi, Mladen Kolar, and Nathan Srebro. Sketching meets random projection in the dual: A provable recovery algorithm for big and high-dimensional data. *arXiv preprint arXiv:1610.03045*, 2016. 5.4.2

[187] Shuohang Wang and Jing Jiang. Machine comprehension using match-lstm and answer pointer. *CoRR*, abs/1608.07905, 2016. URL http://arxiv.org/abs/1608.07905. (document), 2.1, 3.2, 3.7

[188] Wenhui Wang, Nan Yang, Furu Wei, Baobao Chang, and Ming Zhou. Gated self-matching networks for reading comprehension and question answering. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, pages 189–198, 2017. (document), 3.2, 3.5.1, 3.2

[189] Yining Wang, Adams Wei Yu, and Aarti Singh. On computationally tractable selection of experiments in measurement-constrained regression models. *Journal of Machine Learning Research*, 18:143:1–143:41, 2017. 1.4

[190] Zhiguo Wang, Haitao Mi, Wael Hamza, and Radu Florian. Multi-perspective context matching for machine comprehension. *CoRR*, abs/1612.04211, 2016. URL http://arxiv.org/abs/1612.04211. (document), 2.1, 3.2, 3.7

[191] Kilian Q. Weinberger and Lawrence K. Saul. Fast solvers and efficient implementations for distance metric learning. In *ICML*, ICML '08, pages 1160–1167, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-205-4. doi: 10.1145/1390156.1390302. URL http://doi.acm.org/10.1145/1390156.1390302. 5.3, 5.5.2

[192] Kilian Q. Weinberger and Lawrence K. Saul. Distance metric learning for large margin nearest neighbor classification. *Journal of Machine Learning Research*, 10:207–244, June 2009. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=1577069.1577078. 5.3, 5.5.2

[193] Dirk Weissenborn, Georg Wiese, and Laura Seiffe. Making neural QA as simple as possible but not simpler. In *Proceedings of the 21st Conference on Computational Natural Language Learning (CoNLL 2017), Vancouver, Canada, August 3-4, 2017*, pages 271–280, 2017. (document), 3.2, 3.3.2, 3.2

[194] Jason Weston, Antoine Bordes, Sumit Chopra, Alexander M Rush, Bart van Merriënboer, Armand Joulin, and Tomas Mikolov. Towards ai-complete question answering: A set of prerequisite toy tasks. *arXiv preprint arXiv:1502.05698*, 2015. 2.1

[195] John Wieting, Jonathan Mallinson, and Kevin Gimpel. Learning paraphrastic sentence embeddings from back-translated bitext. In *EMNLP*, pages 274–285. Association for Computational Linguistics, 2017. 3.4

[196] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992. 2.2, 2.3.2

[197] Ashia C. Wilson, Rebecca Roelofs, Mitchell Stern, Nati Srebro, and Benjamin Recht. The marginal value of adaptive gradient methods in machine learning. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 4151–4161, 2017. 4.2, 4.5.2, 4.5.3

[198] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016. 2.1, 3.4

[199] Lin Xiao and Stephen P. Boyd. Optimal scaling of a gradient method for distributed resource allocation. *Journal of Optimization Theory and Applications*, 129(3):469–488, June 2006. 6.1

[200] Lin Xiao and Tong Zhang. A proximal stochastic gradient method with progressive variance reduction. *SIAM Journal on Optimization*, 24(4):2057–2075, 2014. 5.2, 6.2.1

[201] Lin Xiao, Adams Wei Yu, Qihang Lin, and Weizhu Chen. DSCOVR: randomized primal-dual block coordinate algorithms for asynchronous distributed optimization. *Journal of Machine Learning Research*, 20:43:1–43:58, 2019. 1.3.2

[202] Eric P. Xing, Qirong Ho, Wei Dai, Jin Kyu Kim, Jinliang Wei, Seunghak Lee, Xun Zheng, Pengtao Xie, Abhimanu Kumar, and Yaoliang Yu. Petuum: A new platform for distributed machine learning on big data. *IEEE Transactions on Big Data*, 1(2):49–67, 2015. 6.1, 6.2.2

[203] Caiming Xiong, Victor Zhong, and Richard Socher. Dynamic coattention networks for question answering. *CoRR*, abs/1611.01604, 2016. URL `http://arxiv.org/abs/1611.01604`. (document), 2.1, 3.1, 3.2, 3.3.2, 3.5.1, 3.2

[204] Tianbao Yang. Trading computation for communication: Distributed stochastic dual coordinate ascent. In *Advances in Neural Information Processing Systems (NIPS) 26*, pages 629–637. 2013. 6.2.2

[205] Tianbao Yang, Yu feng Li, Mehrdad Mahdavi, Rong Jin, and Zhi hua Zhou. Nyström method vs random fourier features: A theoretical and empirical comparison. In *NIPS*, pages 485–493, 2012. 5.4.2

[206] Yang You, Igor Gitman, and Boris Ginsburg. Scaling SGD batch size to 32k for imagenet training. *CoRR*, abs/1708.03888, 2017. 4.5.2

[207] Adams Wei Yu, Fatma Kilinç-Karzan, and Jaime G. Carbonell. Saddle points and accelerated perceptron algorithms. In *ICML*, pages 1827–1835, 2014. 1.4, 5.2

[208] Adams Wei Yu, Wanli Ma, Yaoliang Yu, Jaime G. Carbonell, and Suvrit Sra. Efficient Structured Matrix Rank Minimization. In *NIPS*, 2014. 1.4

[209] Adams Wei Yu, Qihang Lin, and Tianbao Yang. Doubly stochastic primal-dual coordinate method for bilinear saddle-point problem. arXiv:1508.03390, 2015. 1.3.2, 5.2, 6.2.2

[210] Adams Wei Yu, Hongrae Lee, and Quoc V. Le. Learning to Skim Text. *ACL*, 2017. 1.2.1, 3.2

[211] Adams Wei Yu, Qihang Lin, Ruslan Salakhutdinov, and Jaime G. Carbonell. Normalized gradient with adaptive stepsize method for deep neural network training. *arXiv:1707.04822*, 2017. 1.3.1

[212] Adams Wei Yu, David Dohan, Minh-Thang Luong, Rui Zhao, Kai Chen, Mohammad Norouzi, and Quoc V. Le. Qanet: Combining local convolution with global self-attention for reading comprehension. *ICLR*, 2018. 1.2.2

[213] Yang Yu, Wei Zhang, Kazi Saidul Hasan, Mo Yu, Bing Xiang, and Bowen Zhou. End-to-end reading comprehension with dynamic answer chunk ranking. *CoRR*, abs/1610.09996, 2016. URL `http://arxiv.org/abs/1610.09996`. (document), 3.2, 3.7

[214] Hyokyun Yun, Hsiang-Fu Yu, Cho-Jui Hsieh, S V N Vishwanathan, and Inderjit Dhillon. NOMAD: Non-locking, stochastic multi-machine algorithm for asynchronous and decentralized matrix completion. In *Proceedings of the VLDB Endowment*, 2014. 6.2

[215] Matei Zaharia, Reynold Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J. Franklin, Ali Ghodsi, Joseph Gonzalez, Scott Shenker, and Ion Stoica. Apache Spark: A unified engine for big data processing. *Communications of the ACM*, 59(11):56–65, 2016. 6.1

[216] Wojciech Zaremba and Ilya Sutskever. Reinforcement learning neural turing machines-revised. *arXiv preprint arXiv:1505.00521*, 2015. 2.2, 2.3.2

[217] Junbei Zhang, Xiao-Dan Zhu, Qian Chen, Li-Rong Dai, Si Wei, and Hui Jiang. Exploring question understanding and adaptation in neural-network-based question answering. *CoRR*, abs/1703.04617, 2017. (document), 3.2, 3.7

[218] Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In *Advances in neural information processing systems*, pages 649–657, 2015. 2.1, 2.1, 2.4.3, 3.2

[219] Yuchen Zhang and Lin Xiao. DiSCO: Distributed optimization for self-concordant empirical loss. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, pages 362–370, Lille, France, 2015. 6.2.2

[220] Yuchen Zhang and Lin Xiao. Stochastic primal-dual coordinate method for regularized empirical risk minimization. In *ICML*, 2015. 5.2, 5.3, 5.4.1, 5.6, 5.6.2, 6.2.1, 6.2.2, 6.3, A.1

[221] Yuchen Zhang, John C Duchi, and Martin J Wainwright. Communication-efficient algorithms for statistical optimization. *Journal of Machine Learning Research*, 14:3321–3363,

2013. 6.8, 6.8

[222] Tuo Zhao, Mo Yu, Yiming Wang, Raman Arora, and Han Liu. Accelerated mini-batch randomized block coordinate descent method. In *NIPS*, pages 3329–3337, 2014. 5.2

[223] Qingyu Zhou, Nan Yang, Furu Wei, Chuanqi Tan, Hangbo Bao, and Ming Zhou. Neural question generation from text: A preliminary study. *CoRR*, abs/1704.01792, 2017. URL `http://arxiv.org/abs/1704.01792`. 3.2