Distributed Machine Learning Framework: New Algorithms and Theoretical Foundation

by

Zhouyuan Huo

BS, Zhejiang University, 2014

Submitted to the Graduate Faculty of

the Swanson School of Engineering in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

University of Pittsburgh

2020

UNIVERSITY OF PITTSBURGH

SWANSON SCHOOL OF ENGINEERING

This dissertation was presented

by

Zhouyuan Huo

It was defended on

April 1, 2020

and approved by

Heng Huang, PhD, John A. Jurenko Endowed Professor, Department of Electrical and Computer Engineering

Liang Zhan, PhD, Assistant Professor, Department of Electrical and Computer Engineering

Zhi-Hong Mao, PhD, Professor, Department of Electrical and Computer Engineering

Wei Gao, PhD, Associate Professor, Department of Electrical and Computer Engineering

Wei Chen, PhD, Associate Professor, The School of Medicine, Department of Pediatrics

Dissertation Director: Heng Huang, PhD, John A. Jurenko Endowed Professor, Department of Electrical and Computer Engineering

Distributed Machine Learning Framework: New Algorithms and Theoretical Foundation

Zhouyuan Huo, PhD

University of Pittsburgh, 2020

Machine learning is gaining fresh momentum, and has helped us to enhance not only many industrial and professional processes but also our everyday living. The recent success of machine learning relies heavily on the surge of big data, big models, and big computing. However, inefficient algorithms restrict the applications of machine learning to big data mining tasks. In terms of big data, serious concerns, such as communication overhead and data privacy, should be rigorously addressed when we train models using large amounts of data located on multiple devices. In terms of the big model, it is still an underexplored research area if a model is too big to train on a single device. To address these challenging problems, this thesis is focusing on designing new large-scale machine learning models, efficiently optimizing and training methods for big data mining, and studying new discoveries in both theory and applications.

For the challenges raised by big data, we proposed several new asynchronous distributed stochastic gradient descent or coordinate descent methods for efficiently solving convex and non-convex problems. We also designed new large-batch training methods for deep learning models to reduce the computation time significantly with better generalization performance. For the challenges raised by the big model, We scaled up the deep learning models by parallelizing the layer-wise computations with a theoretical guarantee, which is the first algorithm breaking the lock of backpropagation such that the large model can be dramatically accelerated.

Table of Contents

List of Tables

List of Figures

Preface

First and foremost, I would like to thank my Ph.D. advisor, Professor Heng Huang. Heng has been an exceptional advisor and I have been very fortunate to do research under his supervision for the six years of Ph.D. journey. Heng is highly self-motivated and I am deeply impressed by his tremendous passion for research. In the past six years, Heng has been always ready to offer me extremely visionary advice and the most generous support. I learned from Heng how to discover a problem, how to define this problem, and how to solve this problem. I feel very lucky that I can start and pursue my research career under the supervision of Professor Heng Huang.

I would also like to thank Professor Zhi-Hong Mao, Professor Wei Gao, Professor Liang Zhan, and Professor Wei Chen for being on my Ph.D. committee. I really appreciate their great advice and guidance for my research direction. I feel honored to get the inspiring instructions from them and am grateful to the committee members for their valuable time and help.

I want to thank every member in the Pitt Data Science Lab. Thanks Dr. Feiping Nie, Dr. Bin Gu and Dr. Feihu Huang for their guidance, insightful discussions and collaboration during my Ph.D. study. I would thank my friends and lab mates: An, De, Xin, Guodong, Xiaoqian, Hongchang, Kamran, Yanfu, Runxue, Wenhan, Shangqian. I feel very lucky to be a member of such a wonderful research group and it was my pleasure working together with you all.

It has been an honor for me to work with many great collaborators in the academia or when I was a summer intern in the industry lab. I would like to thank Zachary Garrett, Jakub Konečný, Brendan McMahan, Peter Kairouz from Google Research and Hao Jiang, Lin Liang, Quanzeng You from Microsoft Research for advising and helping me in the internship projects. I also enjoyed hanging out and collaborations with other interns, Liang Yang and Ziteng Sun.

Finally, I give special thanks to my families, especially parents, grandparents and Qian. Thank you for raising me up and I can't accomplish what I did without your love and

support. Thanks for your understanding and love in the past years. Because of you, I can get over obstacles in my way. Because of you, I never stray too far from the sidewalk.

1.0 Introduction

1.1 Background

The phenomenal progress of machine learning and the explosive growth of big data have been accelerating the trend of world development. While this vision is expected to generate many disruptive business and social benefits, it presents many unprecedented challenges. In terms of model, current deep learning models are not well designed or too small to learn human knowledge. In terms of data, serious concerns such as data privacy should be rigorously addressed when we train models using amounts of data generated on personal devices. In terms of optimization, training giant neural networks or training models on the device cause new difficulties for current optimization algorithms.

The scalability and efficiency have been the notorious bottlenecks of some machine learning models, constraining them from being applied to big data. Large amounts of data can boost machine learning models to obtain remarkable predictive capabilities. When data are distributed across devices, however, training models suffer from heavy computation, slow communication, or lacking convergence guarantee. An efficient distributed optimization method should make sure that models will converge to solutions in the end with fewer computations and faster communication.

To reduce the communication overhead, asynchronous parallel algorithms for stochastic optimization have received huge successes in theory and practice recently. Because there is no need of synchronization between workers, asynchronous methods often have better performance than synchronous methods. [80] proposed the first asynchronous parallel stochastic gradient descent (SGD) algorithm known as Hogwild!. [39] proposed an asynchronous parallel SGD algorithm with the SVRG variance reduction technique.

The growth of neural network depth is one of the most critical factors contributing to the success of deep learning. It has been verified both in practice and in theory that the more significant number of parameters of a neural network can facilitate its learning ability. For example, the current state-of-the-art models in various applications are giant neural

networks: convolutional neural networks with over 1000 layers for image classification, or Transformer networks with 1.6B parameters for natural language understanding. However, such giant neural networks cannot fit in a single training device. Besides, current optimization algorithms cannot make full use of computing resources, wasting much energy, and affecting the environment. It remains a key problem in deep learning that the deep neural networks with millions of or even billions of parameters cannot fit in a single training device, e.g., GPU or TPU. In many applications such as image classification, object detection and language models, If we allocate and train the parameters of a neural network on multiple GPUs, backward locking in backpropagation algorithm becomes the bottleneck of making full use of computing resources, leading to serious loss of money and energy. The backward locking constrains us from updating models in parallel and fully leveraging the computing resources.

The goal of this thesis is to propose new distributed optimization algorithms and provide theoretical foundation to address the challenges in training distributed machine learning problems using model parallelism and data parallelism.

1.2 Contribution

We summarize our contribution as follows:

• We provide the first theoretical analysis on the convergence rate of asynchronous mini-batch gradient descent with variance reduction (AsySVRG) for non-convex optimization. We prove that both methods can converge with a rate of $O(1/T)$ for non-convex optimization, and linear speedup is accessible when we increase the number of workers.

• We address two challenging issues in previous primal-dual distributed optimization methods: firstly, Dis-dfSDCA does not rely on the dual formulation, and can be used to solve the non-convex problem; secondly, Dis-dfSDCA uses asynchronous communication and can be applied on the complicated distributed system where there is straggler problem. We also analyze the convergence rate of our method and prove the linear convergence rate even if the individual functions in objective are non-convex.

• We propose a novel Complete Layer-wise Adaptive Rate Scaling (CLARS) algorithm for large-batch training. We also analyze the convergence rate of the proposed method by introducing a new fine-grained analysis of gradient-based methods. Based on our analysis, we bridge the gap and illustrate the theoretical insights for three popular large-batch training techniques, including linear learning rate scaling, gradual warmup, and layer-wise adaptive rate scaling.

• We decouple the backpropagation algorithm using delayed gradients, and show that the backward locking is removed when we split the networks into multiple modules. Then, we utilize decoupled parallel backpropagation in two stochastic methods and prove that our method guarantees convergence to critical points for the non-convex problem.

• We propose a novel parallel-objective formulation for the objective function of the neural network. After that, we introduce features replay algorithm and prove that it is guaranteed to converge to critical points for the non-convex problem under certain conditions. Finally, we apply our method to training deep convolutional neural networks, and the experimental results show that the proposed method achieves faster convergence, lower memory consumption, and better generalization error than compared methods.

1.3 Notation

Unless specified otherwise, the notations used in this thesis are listed in Table 1.

1.4 Thesis Organization

The rest of the thesis is organized as follows. In Chapter 2, we introduce a new asynchronous mini-batch gradient descent with variance reduction for non-convex optimization. In Chapter 3, we design a new distributed dual-free SDCA algorithm to address two challenging issues in previous primal-dual distributed optimization methods. In Chapter 4, we propose a novel Complete Layer-wise Adaptive Rate Scaling (CLARS) algorithm for large-

Table 1: Notations used in thesis.

f	objective function
w	model parameter
∇f	gradient
n	number of examples
d	number of parameters
T	number of iterations

batch training. In Chapter 5, we decouple the backpropagation algorithm using delayed gradients and parallelize the neural network training if the model is too big. Next in Chapter 6 we propose a features replay algorithm and show that it outperforms compared model parallelism algorithms. Finally, we conclude the thesis in Chapter 7.

2.0 Asynchronous Mini-batch Gradient Descent with Variance Reduction for Non-Convex Optimization

2.1 Motivation

With the boom of data, training machine learning model with large-scale datasets becomes a challenging problem. Basing on batch gradient descent (GD) method, researchers propose stochastic gradient descent (SGD) method or mini-batch gradient descent method to relieve the complexity of computation in each iteration and reduce the total time complexity for optimization [73, 55, 24, 26, 9]. Due to efficiency, SGD method has been widely used to solve different kinds of large-scale machine learning problems, including both convex and non-convex. However, because we use stochastic gradient to approximate full gradient in the process, a decreasing learning rate has to be applied to guarantee convergence, or it is very easy to diverge from the optimal solution. Thus, it leads to a sub-linear convergence rate of $O(1/T)$ on strongly convex problem. Recently, stochastic variance reduced gradient (SVRG) [46] and its variants, such as SAGA[17], m2SGD[51], have gained much attention in stochastic optimization. Through reusing the previously computed first order gradient information, these methods are able to reduce the variance of gradient approximation in the optimization and are proved to have linear convergence rate on strongly convex problem. After that, SVRG is then applied to solve non-convex problem [4, 81], and it is proved to have a faster sub-linear convergence rate of $O(1/T)$. Experiments are conducted on neural networks and their results also validate that it outperforms SGD method for non-convex optimization.

Serial algorithm is not able to make good use of computation resource. Therefore, parallel algorithms are introduced to further speedup the computation task, including synchronous optimization and asynchronous optimization. Because there is no need of synchronization between workers, asynchronous methods often have better performance. Asynchronous parallelism has been successfully applied to speedup many state-of-the-art optimization algorithms, such as SGD [80, 59], stochastic coordinate descent (SCD) [64], SVRG [109] and Du-

alFreeSDCA [40]. There are mainly two kinds of distributed architectures, one is distributed-memory architecture on multiple machines [2, 59, 108, 16, 109] and the other one is shared-memory architecture on a multi-core machine [80, 111, 112]. Deep learning is a typical situation where asynchronous SGD and its variants have gained great success[56, 16, 59, 75]. It is known that deep neural network always has large set of parameters and trains with large-scale datasets.

Recently, asynchronous SVRG method has been implemented and studied on distributed-memory architecture [109] and shared-memory architecture [111]. It is proved that asynchronous SVRG method has linear convergence rate on strongly convex problem. Mini-batch gradient is implemented in the experiments, while it is missing in their proof. Further, there is no theoretical analysis of convergence rate for asynchronous SVRG on non-convex problem yet.

In this chapter, we provide the convergence analysis of asynchronous mini-batch gradient descent with variance reduction method (asySVRG) for non-convex optimization. Two different algorithms and analysis are proposed on two different distributed architectures, one is shared-memory architecture and the other is distributed-memory architecture. The key difference between these two categories lies on that distributed-memory architecture can ensure the atomicity of reading and writing the whole vector of x, while the shared-memory architecture can usually just ensure atomic reading and writing on a single coordinate of x [59]. We implement asySVRG on two different architectures and analyze their convergence rate based on the mini-batch setting. We prove that asySVRG can get convergence rate of $O(1/T)$ on both architectures. Besides, we also prove that linear speedup is accessible when we increase the number of workers until reaching an upper bound.

We list our main contributions as follows:

- We extend asynchronous shared-memory SVRG method to solve non-convex problem. Our Shared-AsySVRG on shared-memory architecture has faster convergence rate than AsySGD. We prove that Shared-AsySVRG has a convergence rate of $O(1/T)$ for non-convex optimization.
- We extend asynchronous distributed-memory SVRG method to solve non-convex problem. Our Distributed-AsySVRG on distributed-memory architecture has faster conver-

gence rate than AsySGD. We prove that Distributed-AsySVRG has a convergence rate of $O(1/T)$ for non-convex optimization.

- Both of Shared-AsySVRG and Distributed-AsySVRG have linear speedup when we increase the number of threads in a shared-memory architecture or workers in a distributed-memory architecture until reaching an upper bound.

2.2 Preliminaries

We consider the following empirical loss minimization problem:

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

where $f(w)$ and $f_i(w)$ are Lipschitz smooth, they are not necessarily convex. In this chapter, we assume both of them are non-convex.

Following the proof in [59, 81, 4] for non-convex optimization, we use the weighted average of the ℓ_2 norm of full gradient $||\nabla f(w)||^2$ as metric to analyze its convergence property. For further analysis, throughout this chapter, we make the following assumptions for problem (2.1). All of them are very common assumptions in the theoretical analysis for asynchronous stochastic gradient descent method.

Assumption 2.2.1. ***Independence:*** *All random samples i are selected randomly and independently to each other.*

Assumption 2.2.2. ***Unbiased Gradient:*** *The stochastic gradient $\nabla f_i(w)$ is unbiased:*

$$\mathbb{E}\left[\nabla f_i(w)\right] = \nabla f(w). \tag{2.2}$$

Assumption 2.2.3. ***Lipschitz Gradient:*** *We say $\nabla f(w)$ is Lipschitz continuous, and it holds:*

$$||\nabla f(w) - \nabla f(v)||_2 \leq L||w - v||_2. \tag{2.3}$$

Throughout, we also assume that the function $\nabla f_i(w)$ is also Lipschitz continuous, so that $||\nabla f_i(w) - \nabla f_i(v)||_2 \leq L||w - v||_2$.

Assumption 2.2.4. ***Maximum Time Delay:*** *Time delay variable τ of parameters in each worker is upper bounded, namely* $\max \tau \leq \Delta$. *In practice, Δ is related with the number of workers.*

2.3 Shared-Memory Architecture

In this section, we propose AsySVRG method for shared-memory architecture, and prove that it converges with rate $O(1/T)$. It is proved that SVRG has a convergence rate of $O(1/T)$ on non-convex problem [81, 4]. In this section, we follow the convergence analysis in [81], and extends it to asynchronous optimization on shared-memory architecture.

2.3.1 Algorithm Description

Following the setting in [59], we define one iteration as a modification on any single component of x in the shared memory. We use x_t^{s+1} to denote the value of parameter x in the shared memory after $(ms+t)$ iterations, and Equation (2.4) represents the update rule of parameter x in iteration t:

$$(w_{t+1}^{s+1})_{k_t} = (w_t^{s+1})_{k_t} - \eta (v_t^{s+1})_{k_t}\,, \tag{2.4}$$

where $k_t \in \{1, ..., d\}$ is a random index of component in $w \in \mathbb{R}^d$, and learning rate η is constant. Descent direction v_t^{s+1} is defined as follows:

$$v_t^{s+1} = \frac{1}{|I_t|} \sum_{i_t \in I_t} \left(\nabla f_{i_t}(\hat{w}_t^{s+1}) - \nabla f_{i_t}(\tilde{w}^s) + \nabla f(\tilde{w}^s) \right), \tag{2.5}$$

where $\tilde{w}^s$ denotes a snapshot of w after every m iterations. i_t denotes the index of a sample, and I_t is index set of mini-batch samples, and mini-batch size is $|I_t|$. The definition of $\hat{w}_t^{s+1}$ follows the analysis in [59], where $\hat{w}_t^{s+1}$ is assumed to be some earlier state of w in the shared memory.

$$\hat{w}_t^{s+1} = w_t^{s+1} - \sum_{j \in J(t)} (w_{j+1}^{s+1} - w_j^{s+1}), \tag{2.6}$$

Algorithm 1 Shared-AsySVRG

Initialize $w^0 \in \mathbb{R}^d$.

for $s = 0, 1, 2, , .., S-1$ **do**

$\tilde{w}^s \leftarrow w^s$;

Compute full gradient $\nabla f(\tilde{w}^s) \leftarrow \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(\tilde{w}^s)$;

Parallel Computation on Multiple Threads

for $t = 0, 1, 2, ..., m-1$ **do**

Randomly select mini-batch I_t from $\{1, ...n\}$;

Compute v_t^{s+1}: $v_t^{s+1} \leftarrow \frac{1}{|I_t|}\sum_{i_t \in I_t}\left(\nabla f_{i_t}(\hat{w}_t^{s+1}) - \nabla f_{i_t}(\tilde{w}^s) + \nabla f(\tilde{w}^s)\right)$;

Randomly select k_t from $\{1, ..., d\}$;

Update $(w_{t+1}^{s+1})_{k_t}$: $(w_{t+1}^{s+1})_{k_t} \leftarrow (w_t^{s+1})_{k_t} - \eta(v_t^{s+1})_{k_t}$;

end for

$w^{s+1} \leftarrow w_m^{s+1}$;

end for

where $J(t) \in \{t-1, ..., t-\Delta\}$ is a subset of previous iterations, Δ is the upper bound of time delay. In Algorithm 1, we summarize the Shared-AsySVRG on shared-memory architecture.

2.3.2 Convergence Analysis

In this section, we prove that our proposed Shared-AsySVRG method has a sub-linear convergence rate of $O(1/T)$ on non-convex problem. Different from AsySGD method, we are able to bound the variance of gradient update v_t^{s+1} because of the variance reduction technique. And it is crucial for our convergence analysis.

Lemma 2.3.1. *As per the definition of the variance reduced gradient v_t^{s+1} in Equation (2.5), we define,*

$$u_t^{s+1} = \frac{1}{|I_t|}\sum_{i_t \in I_t}\left(\nabla f_{i_t}(w_t^{s+1}) - \nabla f_{i_t}(\tilde{w}^s) + \nabla f(\tilde{w}^s)\right). \tag{2.7}$$

We have the following inequality:

$$\sum_{t=0}^{m-1} \mathbb{E}\left[||v_t^{s+1}||^2\right] \leq \frac{2d}{d - 2L^2\Delta^2\eta^2} \sum_{t=0}^{m-1} \mathbb{E}\left[||u_t^{s+1}||^2\right]. \tag{2.8}$$

where $\mathbb{E}\left[||u_t^{s+1}||^2\right]$ is upper bounded following [82]:

$$\mathbb{E}\left[||u_t^{s+1}||^2\right] \leq 2\mathbb{E}\left[||\nabla f(w_t^{s+1})||^2\right] + \frac{2L^2}{b}\mathbb{E}\left[||w_t^{s+1} - \tilde{w}^s||^2\right]. \tag{2.9}$$

Proof of Corollary 2.3.1: As per the definitions of v_t^{s+1} (2.5) and u_t^{s+1} (2.7):

$$\begin{aligned}
\mathbb{E}\left[||v_t^{s+1}||^2\right] &= \mathbb{E}\left[||v_t^{s+1} - u_t^{s+1} + u_t^{s+1}||^2\right] \\
&\leq 2\mathbb{E}\left[||v_t^{s+1} - u_t^{s+1}||^2\right] + 2\mathbb{E}\left[||u_t^{s+1}||^2\right] \\
&= 2\mathbb{E}\left[||\frac{1}{b}\sum_{i_t \in I_t} \nabla f_{i_t}(\hat{w}_{t,i_t}^{s+1}) - \nabla f_{i_t}(w_t^{s+1})||^2\right] + 2\mathbb{E}\left[||u_t^{s+1}||^2\right] \\
&\leq \frac{2L^2}{b}\sum_{i_t \in I_t} \mathbb{E}\left[||\hat{w}_{t,i_t}^{s+1} - w_t^{s+1}||^2\right] + 2\mathbb{E}\left[||u_t^{s+1}||^2\right] \\
&\leq \frac{2L^2}{b}\sum_{i_t \in I_t} \mathbb{E}\left[||\sum_{j \in J(t,i_t)} (w_j^{s+1} - w_{j+1}^{s+1})_{k_j}||^2\right] + 2\mathbb{E}\left[||u_t^{s+1}||^2\right] \\
&\leq \frac{2L^2\Delta\eta^2}{bd}\sum_{i_t \in I_t}\sum_{j \in J(t,i_t)} \mathbb{E}\left[||v_j^{s+1}||^2\right] + 2\mathbb{E}\left[||u_t^{s+1}||^2\right],
\end{aligned} \tag{2.10}$$

where the first, third and last inequality follows from $||a_1 + ... + a_n||^2 \leq n\sum_{i=1}^{n} ||a_i||^2$. Second inequality follows from Lipschitz smoothness of $f(w)$. Then sum over $\mathbb{E}\left[||v_t^{s+1}||^2\right]$ in one epoch, we get the following inequality,

$$\begin{aligned}
\sum_{t=0}^{m-1} \mathbb{E}\left[||v_t^{s+1}||^2\right] &\leq \sum_{t=0}^{m-1}\left[\frac{2L^2\Delta\eta^2}{bd}\sum_{i_t \in I_t}\sum_{j \in J(t,i_t)} \mathbb{E}\left[||v_j^{s+1}||^2\right] + 2\mathbb{E}\left[||u_t^{s+1}||^2\right]\right] \\
&\leq \frac{2L^2\Delta^2\eta^2}{d}\sum_{t=0}^{m-1} \mathbb{E}\left[||v_t^{s+1}||^2\right] + 2\sum_{t=0}^{m-1} \mathbb{E}\left[||u_t^{s+1}||^2\right].
\end{aligned} \tag{2.11}$$

Thus, if $d - 2L^2\Delta^2\eta^2 > 0$, then $||v_t^{s+1}||^2$ is upper bounded by $||u_t^{s+1}||^2$,

$$\sum_{t=0}^{m-1} \mathbb{E}\left[||v_t^{s+1}||^2\right] \leq \frac{2d}{d - 2L^2\Delta^2\eta^2} \sum_{t=0}^{m-1} \mathbb{E}\left[||u_t^{s+1}||^2\right]. \tag{2.12}$$

The result is different from the proof in [82], because our update step is different. □

From Lemma 2.3.1, we know that the variance of v_t^{s+1} goes to zero when we reach the optimal solution if it exists. Thus, we can maintain learning rate as a constant in the optimization. Therefore, our Shared-AsySVRG has a faster convergence rate as follows:

Theorem 2.3.1. *Suppose all assumptions of $f(w)$ satisfy. Let $c_m = 0$, learning rate $\eta > 0$ is constant, $\beta_t = \beta > 0$, $b = |I_t|$ denotes the size of mini-batch samples in each iteration. We define:*

$$c_t = c_{t+1}(1 + \frac{\eta\beta_t}{d} + \frac{4L^2\eta^2}{(d - 2L^2\Delta^2\eta^2)b}) + \frac{4L^2}{(d - 2L^2\Delta^2\eta^2)b}(\frac{L^2\Delta^2\eta^3}{2d} + \frac{\eta^2 L}{2}), \tag{2.13}$$

$$\Gamma_t = \frac{\eta}{2d} - \frac{4}{d - 2L^2\Delta^2\eta^2}(\frac{L^2\Delta^2\eta^3}{2d} + \frac{\eta^2 L}{2} + c_{t+1}\eta^2), \tag{2.14}$$

such that $\Gamma_t > 0$ for $0 \leq t \leq m-1$. Define $\gamma = \min_t \Gamma_t$, and w^* is the optimal solution for non-convex problem. Then, Shared-AsySVRG algorithm has the following convergence rate in iteration T:

$$\frac{1}{T}\sum_{s=0}^{S-1}\sum_{t=0}^{m-1}\mathbb{E}\left[||\nabla f(w_t^{s+1})||^2\right] \leq \frac{\mathbb{E}\left[f(w^0) - f(w^*)\right]}{T\gamma}. \tag{2.15}$$

Proof of Theorem 2.3.1: At first, we derive the upper bound of $\mathbb{E}\left[||w_{t+1}^{s+1} - \tilde{w}^s||^2\right]$:

$$\begin{aligned}
&\mathbb{E}\left[||w_{t+1}^{s+1} - \tilde{w}^s||^2\right] \\
&= \mathbb{E}\left[||w_{t+1}^{s+1} - w_t^{s+1} + w_t^{s+1} - \tilde{w}^s||^2\right] \\
&= \mathbb{E}\left[||w_{t+1}^{s+1} - w_t^{s+1}||^2 + ||w_t^{s+1} - \tilde{w}^s||^2 + 2\left\langle w_{t+1}^{s+1} - w_t^{s+1}, w_t^{s+1} - \tilde{w}^s\right\rangle\right] \\
&= \mathbb{E}\left[\frac{\eta^2}{d}||v_t^{s+1}||^2 + ||w_t^{s+1} - \tilde{w}^s||^2 - \frac{2\eta}{d}\left\langle \frac{1}{b}\sum_{i_t\in I_t}\nabla f(\hat{w}_{t,i_t}^{s+1}), w_t^{s+1} - \tilde{w}^s\right\rangle\right] \\
&\leq \frac{\eta^2}{d}\mathbb{E}\left[||v_t^{s+1}||^2\right] + \frac{2\eta}{d}\mathbb{E}\left[\frac{1}{2\beta_t}||\frac{1}{b}\sum_{i_t\in I_t}\nabla f(\hat{w}_{t,i_t}^{s+1})||^2 + \frac{\beta_t}{2}||w_t^{s+1} - \tilde{w}^s||^2\right] + \mathbb{E}\left[||w_t^{s+1} - \tilde{w}^s||^2\right] \\
&= \frac{\eta^2}{d}\mathbb{E}\left[||v_t^{s+1}||^2\right] + \frac{\eta}{d\beta_t}\mathbb{E}\left[||\frac{1}{b}\sum_{i_t\in I_t}\nabla f(\hat{w}_{t,i_t}^{s+1})||^2\right] + (1 + \frac{\eta\beta_t}{d})\mathbb{E}\left[||w_t^{s+1} - \tilde{w}^s||^2\right],
\end{aligned} \tag{2.16}$$

where the inequality follows from $\langle a, b\rangle \leq \frac{1}{2}(a^2 + b^2)$. Then we know that $\mathbb{E}\left[f(w_{t+1}^{s+1})\right]$ is also upper bounded:

$$\begin{aligned}
\mathbb{E}\left[f(w_{t+1}^{s+1})\right] &\leq \mathbb{E}\left[f(w_t^{s+1}) + \left\langle \nabla f(w_t^{s+1}), w_{t+1}^{s+1} - w_t^{s+1}\right\rangle + \frac{L}{2}||w_{t+1}^{s+1} - w_t^{s+1}||^2\right] \\
&= \mathbb{E}\left[f(w_t^{s+1})\right] - \frac{\eta}{d}\mathbb{E}\left[\left\langle \nabla f(w_t^{s+1}), \frac{1}{b}\sum_{i_t \in I_t} \nabla f(\hat{w}_{t,i_t}^{s+1})\right\rangle\right] + \frac{\eta^2 L}{2d}\mathbb{E}\left[||v_t^{s+1}||^2\right] \\
&= \mathbb{E}\left[f(w_t^{s+1})\right] - \frac{\eta}{2d}\mathbb{E}\Bigg[||\nabla f(w_t^{s+1})||^2 + ||\frac{1}{b}\sum_{i_t \in I_t} \nabla f(\hat{w}_{t,i_t}^{s+1})||^2 \\
&\quad - ||\nabla f(w_t^{s+1}) - \frac{1}{b}\sum_{i_t \in I_t} \nabla f(\hat{w}_{t,i_t}^{s+1})||^2\Bigg] + \frac{\eta^2 L}{2d}\mathbb{E}\left[||v_t^{s+1}||^2\right], \qquad (2.17)
\end{aligned}$$

where the first inequality follows from Lipschitz continuity of $f(w)$.

$$\begin{aligned}
\mathbb{E}\left[||\nabla f(w_t^{s+1}) - \frac{1}{b}\sum_{i_t \in I_t} \nabla f(\hat{w}_{t,i_t}^{s+1})||^2\right] &\leq \frac{L^2}{b}\sum_{i_t \in I_t}\mathbb{E}\left[||w_t^{s+1} - \hat{w}_{t,i_t}^{s+1}||^2\right] \\
&= \frac{L^2}{b}\sum_{i_t \in I_t}\mathbb{E}\left[||\sum_{j \in J(t,i_t)}(w_j^{s+1} - w_{j+1}^{s+1})||^2\right] \\
&\leq \frac{L^2\Delta}{b}\sum_{i_t \in I_t}\sum_{j \in J(t,i_t)}\mathbb{E}\left[||w_j^{s+1} - w_{j+1}^{s+1}||^2\right] \\
&\leq \frac{L^2\Delta\eta^2}{bd}\sum_{i_t \in I_t}\sum_{j \in J(t,i_t)}\mathbb{E}\left[||v_j^{s+1}||^2\right], \qquad (2.18)
\end{aligned}$$

where the first inequality follows from Lipschitz continuity of $f(w)$. Δ denotes the upper bound of time delay. From (2.17) and (2.18), it is to derive the following inequality:

$$\begin{aligned}
\mathbb{E}\left[f(w_{t+1}^{s+1})\right] \leq \mathbb{E}\left[f(w_t^{s+1})\right] &- \frac{\eta}{2d}\mathbb{E}\left[||\nabla f(w_t^{s+1})||^2\right] - \frac{\eta}{2d}\mathbb{E}\left[||\frac{1}{b}\sum_{i_t \in I_t}\nabla f(\hat{w}_{t,i_t}^{s+1})||^2\right] \\
&+ \frac{\eta^2 L}{2d}\mathbb{E}\left[||v_t^{s+1}||^2\right] + \frac{L^2\Delta\eta^3}{2bd^2}\sum_{i_t \in I_t}\sum_{j \in J(t,i_t)}\mathbb{E}\left[||v_j^{s+1}||^2\right]. \qquad (2.19)
\end{aligned}$$

Following the proof in [81], we define Lyapunov function (this nice proof approach was first introduced in [81]): $R_t^{s+1} = \mathbb{E}\left[f(w_t^{s+1}) + c_t||w_t^{s+1} - \tilde{w}^s||^2\right]$. From the definition of Lyapunov function, and inequalities in (2.16) and (2.19):

$$
\begin{aligned}
R_{t+1}^{s+1} &\leq \mathbb{E}\left[f(w_t^{s+1})\right] - \frac{\eta}{2d}\mathbb{E}\left[||\nabla f(w_t^{s+1})||^2\right] - \frac{\eta}{2d}\mathbb{E}\left[||\frac{1}{b}\sum_{i_t\in I_t}\nabla f(\hat{w}_{t,i_t}^{s+1})||^2\right] \\
&+ \frac{\eta^2 L}{2d}\mathbb{E}\left[||v_t^{s+1}||^2\right] + \frac{L^2\Delta\eta^3}{2bd^2}\sum_{i_t\in I_t}\sum_{j\in J(t,i_t)}\mathbb{E}\left[||v_j^{s+1}||^2\right] \\
&+ c_{t+1}\left[\frac{\eta^2}{d}\mathbb{E}\left[||v_t^{s+1}||^2\right] + (1+\frac{\eta\beta_t}{d})\mathbb{E}\left[||w_t^{s+1} - \tilde{w}^s||^2\right] + \frac{\eta}{d\beta_t}\mathbb{E}\left[||\frac{1}{b}\sum_{i_t\in I_t}\nabla f(\hat{w}_{t,i_t}^{s+1})||^2\right]\right] \\
&\leq \mathbb{E}\left[f(w_t^{s+1})\right] - \frac{\eta}{2d}\mathbb{E}\left[||\nabla f(w_t^{s+1})||^2\right] + \frac{L^2\Delta\eta^3}{2bd^2}\sum_{i_t\in I_t}\sum_{j\in J(t,i_t)}\mathbb{E}\left[||v_j^{s+1}||^2\right] \\
&+ (\frac{\eta^2 L}{2d} + \frac{c_{t+1}\eta^2}{d})\mathbb{E}\left[||v_t^{s+1}||^2\right] + c_{t+1}(1+\frac{\eta\beta_t}{d})\mathbb{E}\left[||w_t^{s+1} - \tilde{w}^s||^2\right], \qquad (2.20)
\end{aligned}
$$

where we assume $\frac{1}{2} \geq \frac{c_{t+1}}{\beta_t}$. As per Corollary 2.3.1, we sum up R_{t+1}^{s+1} from $t = 0$ to $m-1$,

$$
\begin{aligned}
\sum_{t=0}^{m-1} R_{t+1}^{s+1} &\leq \sum_{t=0}^{m-1}\Bigg[\mathbb{E}\left[f(w_t^{s+1})\right] - \frac{\eta}{2d}\mathbb{E}\left[||\nabla f(w_t^{s+1})||^2\right] + \frac{L^2\Delta\eta^3}{2bd^2}\sum_{i_t\in I_t}\sum_{j\in J(t,i_t)}\mathbb{E}\left[||v_j^{s+1}||^2\right] \\
&\quad + (\frac{\eta^2 L}{2d} + \frac{c_{t+1}\eta^2}{d})\mathbb{E}\left[||v_t^{s+1}||^2\right] + c_{t+1}(1+\frac{\eta\beta_t}{d})\mathbb{E}\left[||w_t^{s+1} - \tilde{w}^s||^2\right]\Bigg] \\
&\leq \sum_{t=0}^{m-1}\Bigg[\mathbb{E}\left[f(w_t^{s+1})\right] - \frac{\eta}{2d}\mathbb{E}\left[||\nabla f(w_t^{s+1})||^2\right] + c_{t+1}(1+\frac{\eta\beta_t}{d})\mathbb{E}\left[||w_t^{s+1} - \tilde{w}^s||^2\right] \\
&\quad + (\frac{L^2\Delta^2\eta^3}{2d^2} + \frac{\eta^2 L}{2d} + \frac{c_{t+1}\eta^2}{d})\mathbb{E}\left[||v_t^{s+1}||^2\right]\Bigg] \\
&\leq \sum_{t=0}^{m-1}\Bigg[\mathbb{E}\left[f(w_t^{s+1})\right] - \frac{\eta}{2d}\mathbb{E}\left[||\nabla f(w_t^{s+1})||^2\right] + c_{t+1}(1+\frac{\eta\beta_t}{d})\mathbb{E}\left[||w_t^{s+1} - \tilde{w}^s||^2\right] \\
&\quad + \frac{2d}{d - 2L^2\Delta^2\eta^2}(\frac{L^2\Delta^2\eta^3}{2d^2} + \frac{\eta^2 L}{2d} + \frac{c_{t+1}\eta^2}{d})\mathbb{E}\left[||u_t^{s+1}||^2\right]\Bigg] \\
&\leq \sum_{t=0}^{m-1} R_t^{s+1} - \sum_{t=0}^{m-1}\left[\Gamma_t\mathbb{E}\left[||\nabla f(w_t^{s+1})||^2\right]\right], \qquad (2.21)
\end{aligned}
$$

where

$$
c_t = c_{t+1}(1+\frac{\eta\beta_t}{d}) + \frac{4L^2}{(d - 2L^2\Delta^2\eta^2)b}(\frac{L^2\Delta^2\eta^3}{2d} + \frac{\eta^2 L}{2} + c_{t+1}\eta^2), \qquad (2.22)
$$

$$
\Gamma_t = \frac{\eta}{2d} - \frac{4}{d - 2L^2\Delta^2\eta^2}(\frac{L^2\Delta^2\eta^3}{2d} + \frac{\eta^2 L}{2} + c_{t+1}\eta^2). \qquad (2.23)
$$

Setting $c_m = 0$, $\tilde{w}^{s+1} = w_m^{s+1}$, and $\gamma = \min \Gamma_t$, then $R_m^{s+1} = \mathbb{E}\left[f(w_m^{s+1})\right] = \mathbb{E}\left[f(\tilde{w}^{s+1})\right]$ and $R_0^{s+1} = \mathbb{E}\left[f(w_0^{s+1})\right] = \mathbb{E}\left[f(\tilde{w}^{s})\right]$. Thus we can get,

$$\sum_{t=0}^{m-1} \mathbb{E}\left[||\nabla f(w_t^{s+1})||^2\right] \leq \frac{\mathbb{E}\left[f(\tilde{w}^s) - f(\tilde{w}^{s+1})\right]}{\gamma}. \tag{2.24}$$

Summing up all epochs, and define w^0 as initial point and w^* as optimal solution, we have the final inequality:

$$\frac{1}{T}\sum_{s=0}^{S-1}\sum_{t=0}^{m-1} \mathbb{E}\left[||\nabla f(w_t^{s+1})||^2\right] \leq \frac{\mathbb{E}\left[f(w^0) - f(w^*)\right]}{T\gamma}. \tag{2.25}$$

□

Now, we prove that our method has a convergence rate of $O(1/T)$ if problem is non-convex. If we represent γ with known parameters, we have the following theorem.

Theorem 2.3.2. *Suppose all assumptions of $f(w)$ satisfy. Let $\eta = \frac{u_0 b}{Ln^\alpha}$, where $0 < u_0 < 1$ and $0 < \alpha \leq 1$, $\beta = 2L$, $m = \lfloor \frac{dn^\alpha}{6u_0 b} \rfloor$ and T is the number of total iterations. If the maximum time delay Δ satisfies the following condition:*

$$\Delta^2 < \min\{\frac{d}{2u_0 b}, \frac{3d - 28u_0 bd}{28u_0^2 b^2}\}, \tag{2.26}$$

there exists universal constant u_0 and σ, such that it holds $\gamma \geq \frac{\sigma b}{dLn^\alpha}$ and

$$\frac{1}{T}\sum_{s=0}^{S-1}\sum_{t=0}^{m-1} \mathbb{E}\left[||\nabla f(w_t^{s+1})||^2\right] \leq \frac{dLn^\alpha \mathbb{E}\left[f(w^0) - f(w^*)\right]}{bT\sigma}. \tag{2.27}$$

Proof of Theorem 2.3.2: Following the proof in [81], we set $c_m = 0$, $\eta = \frac{u_0 b}{Ln^\alpha}$, $\beta_t = \beta = 2L$, $0 < u_0 < 1$, and $0 < \alpha < 1$.

$$\begin{aligned} \theta &= \frac{\eta\beta}{d} + \frac{4L^2\eta^2}{(d - 2L^2\Delta^2\eta^2)b} \\ &= \frac{2u_0 b}{dn^\alpha} + \frac{4u_0^2 b}{dn^{2\alpha} - 2\Delta^2 u_0^2 b^2} \\ &\leq \frac{6u_0 b}{dn^\alpha}. \end{aligned} \tag{2.28}$$

In the final inequality, we constrain that $dn^{\alpha} \leq dn^{2\alpha} - 2\Delta^2 u_0^2 b^2$, and it is easy to satisfy when n is large. We set $m = \lfloor \frac{dn^{\alpha}}{6u_0 b} \rfloor$, and from the recurrence formula of c_t, we have:

$$\begin{aligned} c_0 &= \frac{2L^2}{(d - 2L^2\Delta^2\eta^2)b} \left(\frac{L^2\Delta^2\eta^3}{d} + \eta^2 L \right) \frac{(1+\theta)^m - 1}{\theta} \\ &= \frac{2L\left(\frac{u_0^3\Delta^2 b^3}{n^{3\alpha}} + \frac{u_0^2 b^2 d}{n^{2\alpha}} \right)}{(d - 2L^2\Delta^2\eta^2)\left(\frac{2u_0 b^2}{dn^{\alpha}} + \frac{4u_0^2 b^2}{dn^{2\alpha} - 2\Delta^2 u_0^2 b^2} \right) d} \left((1+\theta)^m - 1 \right) \\ &\leq \frac{L(u_0 b\Delta^2 + d)}{3d} \left((1+\theta)^m - 1 \right) \\ &\leq \frac{L(u_0 b\Delta^2 + d)}{3d}(e-1). \end{aligned} \tag{2.29}$$

where the final inequality follows from that $(1+\frac{1}{l})^l$ is increasing for $l > 0$, and $\lim_{l\to\infty}(1+\frac{1}{l})^l = e$. From the proof in Theorem 2.3.1, we know that $c_0 \leq \frac{\beta}{2} = L$,thus $\Delta^2 \leq \frac{d}{2u_0 b}$. c_t is decreasing with respect to t, and c_0 is also upper bounded.

$$\begin{aligned} \gamma &= \min_t \Gamma_t \\ &\geq \frac{\eta}{2d} - \frac{4}{d - 2L^2\Delta^2\eta^2}\left(\frac{L^2\Delta^2\eta^3}{2d} + \frac{\eta^2 L}{2} + c_0\eta^2 \right) \\ &\geq \frac{\eta}{2d} - \frac{4n^{\alpha}}{d}\left(\frac{L^2\Delta^2\eta^3}{2d} + \frac{\eta^2 L}{2} + c_0\eta^2 \right) \\ &\geq \left(\frac{1}{2} - \frac{14u_0^2 b^2\Delta^2 + 14u_0 bd}{3d} \right) \frac{\eta}{d} \\ &\geq \frac{\sigma b}{dLn^{\alpha}}. \end{aligned} \tag{2.30}$$

There exists a small value σ, an it is independent of n. The final inequality holds if $\frac{1}{2} > \frac{14u_0^2 b^2\Delta^2 + 14u_0 bd}{3d}$. Above all, if $\Delta^2 < \min\{\frac{d}{2u_0 b}, \frac{3d - 28u_0 bd}{28u_0^2 b^2}\}$, we have the conclusion that,

$$\frac{1}{T}\sum_{s=0}^{S-1}\sum_{t=0}^{m-1} \mathbb{E}\left[||\nabla f(w_t^{s+1})||^2 \right] \leq \frac{dLn^{\alpha}\mathbb{E}\left[f(\tilde{w}^0) - f(\tilde{w}^*) \right]}{T\sigma b}. \tag{2.31}$$

□

In (2.27), we can find out that the convergence rate has nothing to do with maximum time delay Δ, if it is upper bounded. Thus in a specific domain, the negative effect of using stale information of parameter w for approximating gradient evaluation vanishes, and a linear speedup is accessible when we increase the number of threads.

2.4 Distributed-Memory Architecture

In this section, we propose Distributed-AsySVRG algorithm for distributed-memory architecture, and prove that it converges with rate $O(1/T)$ on non-convex problem.

2.4.1 Algorithm Description

In each iteration, parameter w is updated through the following update rule,

$$w_{t+1}^{s+1} = w_t^{s+1} - \eta v_t^{s+1}, \tag{2.32}$$

where learning rate η is constant, v_t^{s+1} represents variance reduced gradient, and it is defined as follows:

$$v_t^{s+1} = \frac{1}{|I_t|} \sum_{i_t \in I_t} \left(\nabla f_{i_t}(w_{t-\tau}^{s+1}) - \nabla f_{i_t}(\tilde{w}^s) + \nabla f(\tilde{w}^s) \right), \tag{2.33}$$

where $\tilde{w}^s$ means a snapshot of w after every m iterations, and $w_{t-\tau}^{s+1}$ denotes the current parameter used to compute gradient in the worker. i_t denotes the index of a sample, τ denotes time delay of parameter in the worker, and mini-batch size is $|I_t|$. Suppose there are K workers in total, and the number of dataset in worker k is n_k. We summarize the Distributed-AsySVRG on distributed-memory architecture in the Algorithm 2 and Algorithm 3, Algorithm 2 shows operations in server node, and Algorithm 3 shows operations in worker node.

2.4.2 Convergence Analysis

Similar to the convergence analysis in Shared-AsySVRG, we analyze the convergence rate for our proposed Distributed-AsySVRG in this section. It has been proved in [82] that the variance of v_t^{s+1} is upper bounded, and it goes to zero when w is close to the optimal solution.

Algorithm 2 Distributed-AsySVRG Server Node

Initialize $w^0 \in \mathbb{R}^d$.

for $s = 0, 1, 2, , .., S-1$ **do**

$\tilde{w}^s \leftarrow w^s$;

Broadcast $\tilde{w}^s$ to all workers;

Receive and compute: $\nabla f(\tilde{w}^s) \leftarrow \frac{1}{n} \sum_{k=1}^{K} \nabla^k f(\tilde{w}^s)$;

Broadcast $\nabla f(\tilde{w}^s)$ to all workers;

for $t = 0, 1, 2, ..., m-1$ **do**

Receive variance reduced gradient v_t^{s+1} from worker;

Update $w_{t+1}^{s+1} \leftarrow w_t^{s+1} - \eta v_t^{s+1}$;

end for

$w^{s+1} \leftarrow w_m^{s+1}$;

end for

Theorem 2.4.1. *Suppose all assumptions of $f(w)$ satisfy. Let $c_m = 0$, learning rate $\eta > 0$ is constant, $\beta_t = \beta > 0$, b denotes the size of mini-batch. We define the following equations regarding c_t as follows:*

$$c_t = c_{t+1}\left(1 + \eta\beta_t + \frac{4L^2\eta^2}{(1-2L^2\Delta^2\eta^2)b}\right) + \frac{4L^2}{(1-2L^2\Delta^2\eta^2)b}\left(\frac{L^2\Delta^2\eta^3}{2} + \frac{\eta^2 L}{2}\right), \tag{2.34}$$

and Γ_t as follows:

$$\Gamma_t = \frac{\eta}{2} - \frac{4}{(1-2L^2\Delta^2\eta^2)}\left(\frac{L^2\Delta^2\eta^3}{2} + \frac{\eta^2 L}{2} + c_{t+1}\eta^2\right), \tag{2.35}$$

where the requirements are satisfied that $\Gamma_t > 0$ for $0 \leq t \leq m-1$. Define $\gamma = \min_t \Gamma_t$, w^* is the optimal solution for non-convex problem. Then, we have the following convergence rate in iteration T:

$$\frac{1}{T}\sum_{s=0}^{S-1}\sum_{t=0}^{m-1} \mathbb{E}\left[||\nabla f(w_t^{s+1})||^2\right] \leq \frac{\mathbb{E}\left[f(w^0) - f(w^*)\right]}{T\gamma}. \tag{2.36}$$

Algorithm 3 Distributed-AsySVRG Worker Node k

if flag is True **then**

Receive parameter $\tilde{w}^s$ from server;

Compute and send full gradient $\nabla^k f(\tilde{w}^s)$:

$\nabla^k f(\tilde{w}^s) = \sum_{i=1}^{n_k} \nabla_i f(\tilde{w}^s)$;

Receive full gradient $\nabla f(\tilde{w}^s)$ from server;

else

Receive parameter $w_{t-\tau}^{s+1}$ from server;

Randomly select mini-batch I_t from $\{1, ..., n_k\}$;

Compute v_t^{s+1} and send it to server:

$v_t^{s+1} \leftarrow \frac{1}{|I_t|} \sum_{i_t \in I_t} \left(\nabla f_{i_t}(w_{t-\tau}^{s+1}) - \nabla f_{i_t}(\tilde{w}^s) + \nabla f(\tilde{w}^s) \right)$;

end if

Proof of Theorem 2.4.1:

$$
\begin{aligned}
&\mathbb{E}\left[||w_{t+1}^{s+1} - \tilde{w}^s||^2\right] = \mathbb{E}\left[||w_{t+1}^{s+1} - w_t^{s+1} + w_t^{s+1} - \tilde{w}^s||^2\right] \\
&= \mathbb{E}\left[||w_{t+1}^{s+1} - w_t^{s+1}||^2 + ||w_t^{s+1} - \tilde{w}^s||^2 + 2\left\langle w_{t+1}^{s+1} - w_t^{s+1}, w_t^{s+1} - \tilde{w}^s \right\rangle\right] \\
&= \mathbb{E}\left[\eta^2 ||v_t^{s+1}||^2 + ||w_t^{s+1} - \tilde{w}^s||^2 - 2\eta \left\langle \frac{1}{b} \sum_{i_t \in I_t} \nabla f(w_{t-\tau_i}^{s+1}), w_t^{s+1} - \tilde{w}^s \right\rangle\right] \\
&\leq \eta^2 \mathbb{E}\left[||v_t^{s+1}||^2\right] + 2\eta \mathbb{E}\left[\frac{1}{2\beta_t} ||\frac{1}{b} \sum_{i_t \in I_t} \nabla f(w_{t-\tau_i}^{s+1})||^2 + \frac{\beta_t}{2} ||w_t^{s+1} - \tilde{w}^s||^2\right] \\
&+ \mathbb{E}\left[||w_t^{s+1} - \tilde{w}^s||^2\right] \\
&= \eta^2 \mathbb{E}\left[||v_t^{s+1}||^2\right] + (1 + \eta\beta_t) \mathbb{E}\left[||w_t^{s+1} - \tilde{w}^s||^2\right] + \frac{\eta}{\beta_t} \mathbb{E}\left[||\frac{1}{b} \sum_{i_t \in I_t} \nabla f(w_{t-\tau_i}^{s+1})||^2\right], \qquad (2.37)
\end{aligned}
$$

where the first inequality follows $2\langle a, b\rangle \leq ||a||^2 + ||b||^2$.

$$
\begin{aligned}
&\mathbb{E}\left[f(w_{t+1}^{s+1})\right] \leq \mathbb{E}\left[f(w_t^{s+1}) + \left\langle \nabla f(w_t^{s+1}), w_{t+1}^{s+1} - w_t^{s+1} \right\rangle + \frac{L}{2} ||w_{t+1}^{s+1} - w_t^{s+1}||^2\right] \\
&= -\frac{\eta}{2} \mathbb{E}\left[||\nabla f(w_t^{s+1})||^2 + ||\frac{1}{b} \sum_{i_t \in I_t} \nabla f(w_{t-\tau_i}^{s+1})||^2 - ||\nabla f(w_t^{s+1}) - \frac{1}{b} \sum_{i_t \in I_t} \nabla f(w_{t-\tau_i}^{s+1})||^2\right] \\
&+ \mathbb{E}\left[f(w_t^{s+1})\right] + \frac{\eta^2 L}{2} \mathbb{E}\left[||v_t^{s+1}||^2\right], \qquad (2.38)
\end{aligned}
$$

where the first inequality follows from Lipschitz continuity of $f(w)$. Then we know the upper bound of $||\nabla f(w_t^{s+1}) - \frac{1}{b}\sum_{i_t \in I_t} \nabla f(w_{t-\tau_i}^{s+1})||^2$ as follows:

$$\begin{aligned} ||\nabla f(w_t^{s+1}) - \frac{1}{b}\sum_{i_t \in I_t} \nabla f(w_{t-\tau_i}^{s+1})||^2 &\leq \frac{1}{b}\sum_{i_t \in I_t} ||\nabla f(w_t^{s+1}) - \nabla f(w_{t-\tau_i}^{s+1})||^2 \\ &\leq \frac{L^2}{b}\sum_{i_t \in I_t} ||w_t^{s+1} - w_{t-\tau_i}^{s+1}||^2 \\ &\leq \frac{L^2\Delta}{b}\sum_{i_t \in I_t}\sum_{j=t-\tau_i}^{t-1} ||w_j^{s+1} - w_{j+1}^{s+1}||^2 \\ &= \frac{L^2\Delta\eta^2}{b}\sum_{i_t \in I_t}\sum_{j=t-\tau_i}^{t-1} ||v_j^{s+1}||^2, \end{aligned} \tag{2.39}$$

where the second inequality follows from Lipschitz continuity of $f(w)$. Δ denotes the upper bound of time delay. $\tau \leq \Delta$. Above all, we have the following inequality,

$$\begin{aligned} \mathbb{E}\left[f(w_{t+1}^{s+1})\right] \leq \mathbb{E}\left[f(w_t^{s+1})\right] - \frac{\eta}{2}\mathbb{E}\left[||\nabla f(w_t^{s+1})||^2\right] - \frac{\eta}{2}\mathbb{E}\left[||\frac{1}{b}\sum_{i \in I_t}\nabla f(w_{t-\tau_i}^{s+1})||^2\right] \\ + \frac{\eta^2 L}{2}\mathbb{E}\left[||v_t^{s+1}||^2\right] + \frac{L^2\Delta\eta^3}{2b}\sum_{i \in I_t}\sum_{j=t-\tau_i}^{t-1}\mathbb{E}\left[||v_j^{s+1}||^2\right]. \end{aligned} \tag{2.40}$$

Following the definition of R_{t+1}^{s+1},

$$\begin{aligned} R_{t+1}^{s+1} &= \mathbb{E}\left[f(w_{t+1}^{s+1}) + c_{t+1}||w_{t+1}^{s+1} - \tilde{w}^s||^2\right] \\ &\leq \mathbb{E}\left[f(w_t^{s+1})\right] - \frac{\eta}{2}\mathbb{E}\left[||\nabla f(w_t^{s+1})||^2\right] - \frac{\eta}{2}\mathbb{E}\left[||\frac{1}{b}\sum_{i \in I_t}\nabla f(w_{t-\tau_i}^{s+1})||^2\right] \\ &+ \frac{\eta^2 L}{2}\mathbb{E}\left[||v_t^{s+1}||^2\right] + \frac{L^2\Delta\eta^3}{2b}\sum_{i \in I_t}\sum_{j=t-\tau_i}^{t-1}\mathbb{E}\left[||v_j^{s+1}||^2\right] \\ &+ c_{t+1}\left[\eta^2\mathbb{E}\left[||v_t^{s+1}||^2\right] + (1+\eta\beta_t)\mathbb{E}\left[||w_t^{s+1} - \tilde{w}^s||^2\right] + \frac{\eta}{\beta_t}\mathbb{E}\left[||\frac{1}{b}\sum_{i \in I_t}\nabla f(w_{t-\tau_i}^{s+1})||^2\right]\right] \\ &\leq \mathbb{E}\left[f(w_t^{s+1})\right] - \frac{\eta}{2}\mathbb{E}\left[||\nabla f(w_t^{s+1})||^2\right] + \frac{L^2\Delta\eta^3}{2b}\sum_{i \in I_t}\sum_{j=t-\tau_i}^{t-1}\mathbb{E}\left[||v_j^{s+1}||^2\right] \\ &+ (\frac{\eta^2 L}{2} + c_{t+1}\eta^2)\mathbb{E}\left[||v_t^{s+1}||^2\right] + c_{t+1}(1+\eta\beta_t)\mathbb{E}\left[||w_t^{s+1} - \tilde{w}^s||^2\right]. \end{aligned} \tag{2.41}$$

In the final inequality, we make $(\frac{\eta}{2} - \frac{c_{t+1}\eta}{\beta_t}) > 0$. Then we sum over R_{t+1}^{s+1}, the following inequality holds that:

$$\begin{aligned}
&\sum_{t=0}^{m-1} R_{t+1}^{s+1} \leq \sum_{t=0}^{m-1}\Bigg[\mathbb{E}\left[f(w_t^{s+1})\right] - \frac{\eta}{2}\mathbb{E}\left[||\nabla f(w_t^{s+1})||^2\right] + \frac{L^2\Delta\eta^3}{2b}\sum_{i\in I_t}\sum_{j=t-\tau_i}^{t-1}\mathbb{E}\left[||v_j^{s+1}||^2\right] \\
&+ (\frac{\eta^2 L}{2} + c_{t+1}\eta^2)\mathbb{E}\left[||v_t^{s+1}||^2\right] + c_{t+1}(1+\eta\beta_t)\mathbb{E}\left[||w_t^{s+1} - \tilde{w}^s||^2\right]\Bigg] \\
&\leq \sum_{t=0}^{m-1}\Bigg[\mathbb{E}\left[f(w_t^{s+1})\right] - \frac{\eta}{2}\mathbb{E}\left[||\nabla f(w_t^{s+1})||^2\right] + c_{t+1}(1+\eta\beta_t)\mathbb{E}\left[||w_t^{s+1} - \tilde{w}^s||^2\right] \\
&+ (\frac{L^2\Delta^2\eta^3}{2} + \frac{\eta^2 L}{2} + c_{t+1}\eta^2)\mathbb{E}\left[||v_t^{s+1}||^2\right]\Bigg] \\
&\leq \sum_{t=0}^{m-1}\Bigg[\mathbb{E}\left[f(w_t^{s+1})\right] - \frac{\eta}{2}\mathbb{E}\left[||\nabla f(w_t^{s+1})||^2\right] + c_{t+1}(1+\eta\beta_t)\mathbb{E}\left[||w_t^{s+1} - \tilde{w}^s||^2\right] \\
&+ \frac{2}{1-2L^2\Delta^2\eta^2}(\frac{L^2\Delta^2\eta^3}{2} + \frac{\eta^2 L}{2} + c_{t+1}\eta^2)\mathbb{E}\left[||u_t^{s+1}||^2\right]\Bigg] \\
&= \sum_{t=0}^{m-1} R_t^{s+1} - \sum_{t=0}^{m-1}\left[\Gamma_t\mathbb{E}\left[||\nabla f(w_t^{s+1})||^2\right]\right], \qquad (2.42)
\end{aligned}$$

where the last inequality follows the upper bound of v_t^{s+1} in [82], and we define:

$$c_t = c_{t+1}\left(1 + \eta\beta_t + \frac{4L^2\eta^2}{(1-2L^2\Delta^2\eta^2)b}\right) + \frac{4L^2}{(1-2L^2\Delta^2\eta^2)b}\left(\frac{L^2\Delta^2\eta^3}{2} + \frac{\eta^2 L}{2}\right) \qquad (2.43)$$

$$\Gamma_t = \frac{\eta}{2} - \frac{4}{(1-2L^2\Delta^2\eta^2)}(\frac{L^2\Delta^2\eta^3}{2} + \frac{\eta^2 L}{2} + c_{t+1}\eta^2). \qquad (2.44)$$

We set $c_m = 0$, and $\tilde{w}^{s+1} = w_m^{s+1}$, and $\gamma = \min_t \Gamma_t$, thus $R_m^{s+1} = \mathbb{E}\left[f(w_m^{s+1})\right] = \mathbb{E}\left[f(\tilde{w}^{s+1})\right]$, and $R_0^{s+1} = \mathbb{E}\left[f(w_0^{s+1})\right] = \mathbb{E}\left[f(\tilde{w}^s)\right]$. Summing up all epochs, the following inequality holds,

$$\frac{1}{T}\sum_{s=0}^{S-1}\sum_{t=0}^{m-1}\mathbb{E}\left[||\nabla f(w_t^{s+1})||^2\right] \leq \frac{\mathbb{E}\left[f(w^0) - f(w^*)\right]}{T\gamma}. \qquad (2.45)$$

□

Theorem 2.4.2. *Suppose all assumptions of $f(w)$ satisfy. Let $\eta_t = \eta = \frac{u_0 b}{Ln^\alpha}$, where $0 < u_0 < 1$ and $0 < \alpha \leq 1$, $\beta = 2L$, $m = \lfloor \frac{n^\alpha}{6u_0 b} \rfloor$ and T is total iteration. If the maximum time delay Δ is upper bounded by:*

$$\Delta^2 < \min\{\frac{1}{2u_0 b}, \frac{3 - 28u_0 b}{28u_0^2 b^2}\}, \tag{2.46}$$

there exists universal constant u_0, σ, such that if it holds $\gamma \geq \frac{\sigma b}{Ln^\alpha}$, we have the following inequality:

$$\frac{1}{T}\sum_{s=0}^{S-1}\sum_{t=0}^{m-1}\mathbb{E}\left[||\nabla f(w_t^{s+1})||^2\right] \leq \frac{Ln^\alpha \mathbb{E}\left[f(w^0) - f(w^*)\right]}{bT\sigma}. \tag{2.47}$$

Proof of Theorem 2.4.2. Following the proof of Theorem 2.4.1, we let $c_m = 0$, $\eta_t = \eta = \frac{u_0 b}{Ln^\alpha}$, $\beta_t = \beta = 2L$, $0 < u_0 < 1$, and $0 < \alpha < 1$. We define θ, and get its upper bound,

$$\begin{aligned}
\theta &= \eta\beta + \frac{4L^2\eta^2}{(1 - 2L^2\Delta^2\eta^2)b} \\
&= \frac{2u_0 b}{n^\alpha} + \frac{4u_0^2 b}{n^{2\alpha} - 2\Delta^2 u_0^2 b^2} \\
&\leq \frac{6u_0 b}{n^\alpha},
\end{aligned} \tag{2.48}$$

where we assume $n^{2\alpha} - 2\Delta^2 u_0^2 b^2 \geq n^\alpha$. We set $m = \lfloor \frac{n^\alpha}{6u_0 b} \rfloor$, from the recurrence formula between c_t and c_{t+1}, c_0 is upper bounded,

$$\begin{aligned}
c_0 &= \frac{2L^2}{(1 - 2L^2\Delta^2\eta^2)b}\left(L^2\Delta^2\eta^3 + \eta^2 L\right)\frac{(1+\theta)^m - 1}{\theta} \\
&\leq \frac{2L\left(\frac{u_0^3\Delta^2 b^3}{n^{3\alpha}} + \frac{u_0^2 b^2}{n^{2\alpha}}\right)}{(1 - 2L^2\Delta^2\eta^2)\left(\frac{2u_0 b^2}{n^\alpha} + \frac{4u_0^2 b^2}{n^{2\alpha} - 2\Delta^2 u_0^2 b^2}\right)}\left((1+\theta)^m - 1\right) \\
&\leq \frac{L(u_0 b\Delta^2 + 1)}{3}\left((1+\theta)^m - 1\right) \\
&\leq \frac{L(u_0 b\Delta^2 + 1)}{3}(e - 1).
\end{aligned} \tag{2.49}$$

where the final inequality follows from that $(1+\frac{1}{l})^l$ is increasing for $l > 0$, and $\lim_{l\to\infty}(1+\frac{1}{l})^l = e$. From Theorem 2.4.1, we know that $c_0 < \frac{\beta}{2} = L$, then $u_0 b\Delta^2 < \frac{1}{2}$. c_t is decreasing with respect to t, and c_0 is also upper bounded. Now, we can get a lower bound of γ,

$$
\begin{aligned}
\gamma &= \min_t \Gamma_t \\
&\geq \frac{\eta}{2} - \frac{4}{(1-2L^2\Delta^2\eta^2)}(\frac{L^2\Delta^2\eta^3}{2} + \frac{\eta^2 L}{2} + c_0\eta^2) \\
&\geq \frac{\eta}{2} - 4n^\alpha(\frac{L^2\Delta^2\eta^3}{2} + \frac{\eta^2 L}{2} + c_0\eta^2) \\
&\geq (\frac{1}{2} - \frac{14\Delta^2 u_0^2 b^2 + 14u_0 b}{3})\eta \\
&\geq \frac{\sigma b}{Ln^\alpha}.
\end{aligned} \tag{2.50}
$$

There exists a small value σ that the final inequality holds if $\frac{1}{2} > \frac{14\Delta^2 u_0^2 b^2 + 14u_0 b}{3}$. So, if Δ^2 has an upper bound $\Delta^2 < \min\{\frac{1}{2u_0 b}, \frac{3-28u_0 b}{28u_0^2 b^2}\}$, we can prove the final conclusion,

$$
\frac{1}{T}\sum_{s=0}^{S-1}\sum_{t=0}^{m-1}\mathbb{E}\left[||\nabla f(x_t^{s+1})||^2\right] \leq \frac{Ln^\alpha \mathbb{E}\left[f(\tilde{x}^0) - f(\tilde{x}^*)\right]}{bT\sigma}. \tag{2.51}
$$

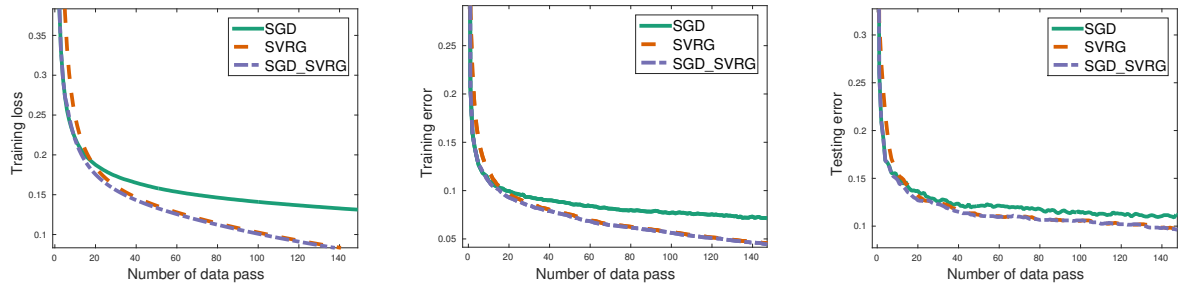
□

Therefore, it is obvious that our proposed Distributed-AsySVRG method has sub-linear convergence rate of $O(1/T)$, and is much faster than the AsySGD with convergence rate of $O(1/\sqrt{T})$ [59]. From inequality (2.47), we know that the convergence rate has nothing to do with Δ if it is upper bounded, linear speedup is also accessible when we increase the number of workers in a cluster.

2.5 Experimental Results

In this section, we perform experiments on shared-memory architecture and distributed-memory architecture respectively. One of the main purposes of our experiments is to validate the faster convergence rate of asySVRG method, and the other purpose is to demonstrate its linear speedup property. The speedup we consider in this chapter is running time speedup when they reach similar performance, e.g. similar training loss function value. Given K workers, running time speedup is defined as,

$$
\text{Time speedup} = \frac{\text{Running time for the serial computation}}{\text{Running time of using } K \text{ workers}}. \tag{2.52}
$$



(a) (b) (c)

Figure 1: Comparison of three methods: SGD, SVRG, SGDSVRG on MNIST dataset.

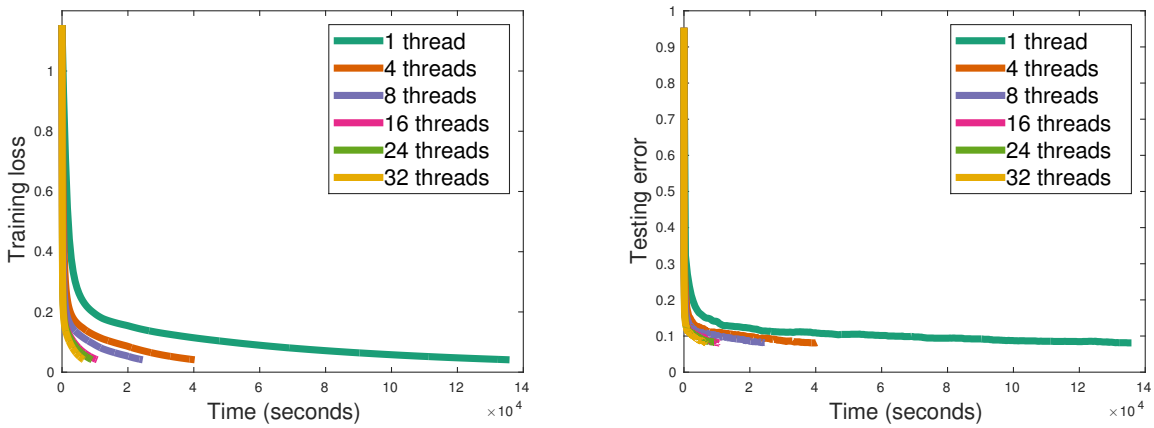
2.5.1 Shared-Memory Architecture

We conduct experiments on a machine which has 2 sockets, and each socket has 18 cores. OpenMP library [1] is used to handle shared-memory parallelism. We consider the multi-class classification task on MNIST dataset [57], and use $10,000$ training samples and $2,000$ testing samples in the experiment. Each image sample is a vector of 784 pixels. We construct a toy three-layer neural network ($784 \times 100 \times 10$), where ReLU activation function is used in the hidden layer. We train this neural network with softmax loss function, and ℓ_2 regularization with weight $C = 10^{-3}$. We set mini-batch size $|I_t| = 10$, and inner iteration length $m = 1,000$. Updating only one component of w in each iteration is too time consuming, therefore we randomly select and update $1,000$ components.

We compare following three methods in the experiment:

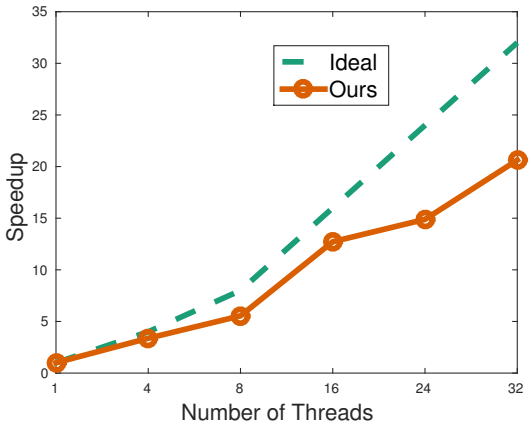
- SGD: We implement stochastic gradient descent (SGD) algorithm and train with the best tuned learning rate. In our experiment, we use polynomial learning rate $\eta = \frac{\alpha}{(1+s)^\beta}$, where α is tuned from $\{1e^{-2}, 5e^{-2}, 1e^{-3}, 5e^{-3}, 1e^{-4}, 5e^{-4}, 1e^{-5}, 5e^{-5}\}$, β is tuned from in $\{0, 0.1, 0.2, ..., 1\}$ and s denotes the epoch number.
- SVRG: We implement our Shared-AsySVRG method and train with the best tuned constant learning rate α.

[1] https://openmp.org



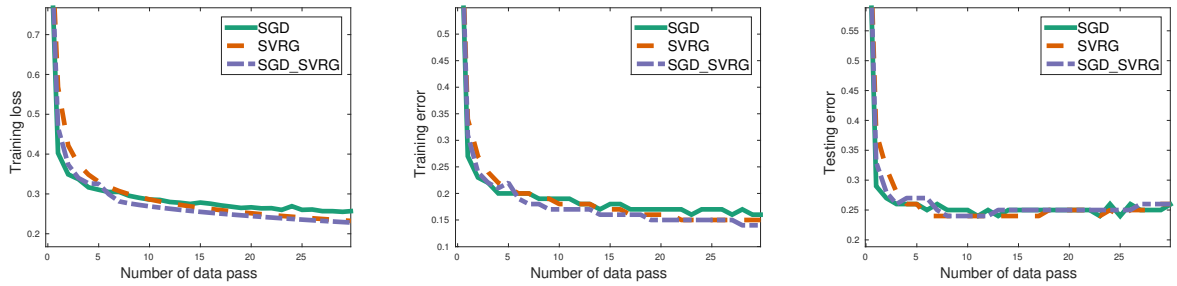
(a)

(b)



(c)

Figure 2: Speedup of Shared-AsySVRG on a machine with different number of threads from 1 to 32.



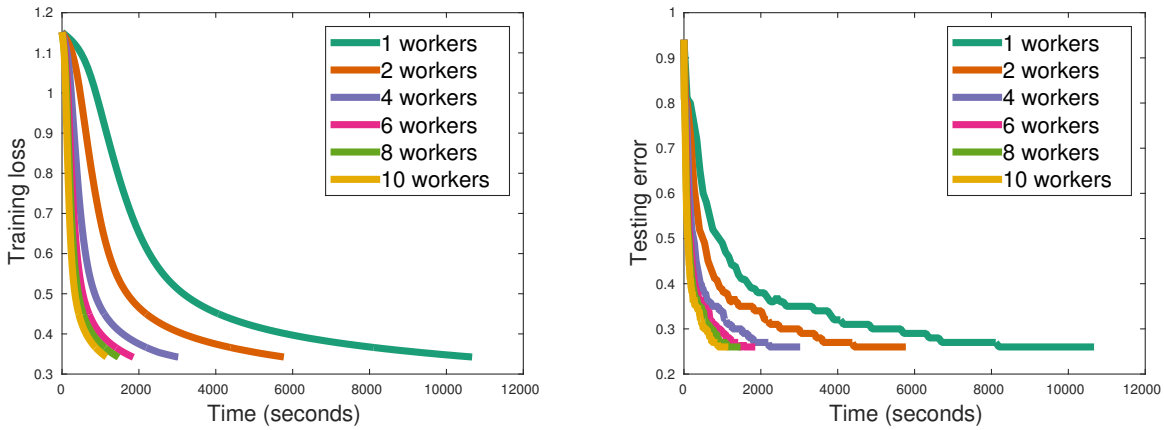
(a) (b) (c)

Figure 3: Comparison of three methods: SGD, SVRG, SGDSVRG on CIFAR-10.

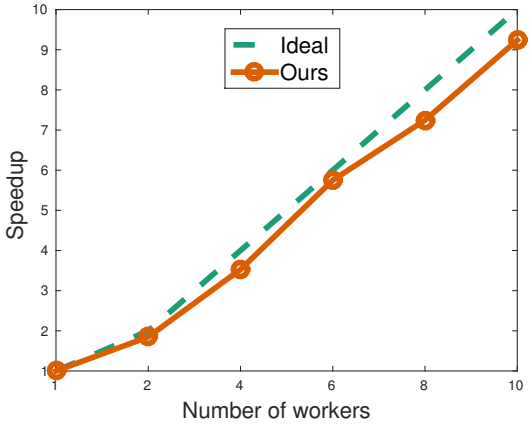
- SGD_SVRG: SVRG method is sensitive to initial point, and it is slower than SGD at first few iterations. Thus, we apply Shared-AsySVRG on a pre-trained model learned by SGD. In the experiment, we use a pre-trained model after running 10 epochs of SGD method.

We evaluate three compared methods on MNIST dataset, and each method trains with the best tuned learning rate. Figure 1 shows the convergence of each method with respect to different criterion: loss function value on training dataset, training error, and testing error. Figure 1a shows the curves of training loss function value, it is clear that SGD method converges faster than SVRG method in the first 20 iterations, and after that, SVRG method outperforms SGD. SGD_SVRG method initializes with a pre-trained model, and it has the best performance. Figure 1b and Figure 1c present the performance of each method on training error and testing error respectively. We can conclude that SVRG and SGD_SVRG method have better performance on the long run, and SGD_SVRG method has the fastest convergence.

To demonstrate that our proposed Shared-AsySVRG method has linear speedup when we increase the number of workers, we also evaluate Shared-AsySVRG with different number of threads, and Figure 2 presents the result of our experiment. In Figure 2a, all curves are reaching the similar training loss value. As we can see, the more threads we use in the



(a) (b)



(c)

Figure 4: Speedup of Distributed-AsySVRG on multiple machines from 1 to 10.

computation, the less time we need to achieve a similar accuracy. This result is reasonable, because when we distribute the whole work to multiple workers, each worker focuses on its own subset independently and parallelly. The ideal result of parallel computation is linear speedup, namely if we use K threads, its running time should be $\frac{1}{K}$ of the time when we just use a single thread. Figure 2c shows the ideal speedup and actual speedup in our experiment. We can find out that a nearly linear speedup is accessible when we increase the thread number. When the number of threads exceeds a threshold, performance will degrade. These findings in the experiment are compatible with our theoretical analysis.

2.5.2 Distributed-Memory Architecture

We conduct distributed-memory architecture experiment on AWS platform[2], and each node is a t2.micro instance with one virtual CPU. Each server and worker takes a single node. The point to point communication between server and workers are handled by MPICH library[3]. We use CIFAR-10 dataset [53] in the experiment, and this dataset has 10 classes of color image of size $32 \times 32 \times 3$. We use $20,000$ samples as training data and $4,000$ samples as testing data. We use a pre-trained CNN model in TensorFlow tutorial [1], and extract features from second fully connected layer. Thus, each sample is a vector of size 384. We construct a three-layer fully connected neural network ($384 \times 50 \times 10$). In the hidden layer, we use ReLU activation function. We train this model with softmax loss, and ℓ_2 regularization with weight $C = 1e^{-4}$. In this experiment, mini-batch size $|I_t| = 10$, and the inner loop length $m = 2,000$. We use the same compared methods as in the last section, except that SGD_SVRG method is initialized with parameters learned after 1 epoch of SGD.

Performances of all three methods are presented in Figure 3. Curves in Figure 3a show that SGD is the fastest method in the first few iterations, after that, SVRG-based method will outperform it. It is obvious that SGD_SVRG has better convergence rate than SVRG method. We can also draw a similar conclusion from Figure 3b. In Figure 3c, it shows that the test error performance of three compared methods are comparable. We also test our Distributed-AsySVRG method with different number of workers, and Figure 4 illustrates

[2]https://aws.amazon.com/
[3]http://www.mpich.org/

the results of our experiment. It is easy to know that when the number of workers increases, our method has a near linear speedup.

3.0 Asynchronous Dual Free Stochastic Dual Coordinate Ascent for Distributed Data Mining

3.1 Motivation

In this chapter, we consider solving the ℓ_2-norm regularized empirical loss minimization problem which is arising ubiquitously in supervised machine learning and data mining problems:

$$\min_{w\in\mathbb{R}^d} P(w) := \min_{w\in\mathbb{R}^d} \frac{1}{n}\sum_{i=1}^{n}\phi_i(w) + \frac{\lambda}{2}\|w\|_2^2. \tag{3.1}$$

We let $f(w) = \frac{1}{n}\sum_{i=1}^{n}\phi_i(w)$ and $w \in \mathbb{R}^d$ be the linear predictor to be optimized. There are many applications falling into this formulation, such as classification, regression, and principal component analysis (PCA). In classification, given features $x_i \in \mathbb{R}^d$ and labels $y_i \in \{1, -1\}$, we obtain Support Vector Machine (SVM) when we let $\phi_i(w) = \max\{0, 1 - y_i x_i^T w\}$. In regression, given features $x_i \in \mathbb{R}^d$ and response $y_i \in \mathbb{R}$, we have Ridge Regression problem if $\phi_i(w) = (y_i - x_i^T w)^2$. Recently, [23, 5] showed that the problem of PCA can be solved through convex optimization. Supposing $C = \frac{1}{n}\sum_{i=1}^{n} x_i x_i^T$ be normalized covariance matrix, [23] showed that approximating the principle component of A is equivalent to minimizing $f(w) = \frac{1}{2}w^T(\mu I - C)w - b^T w$ given $\mu > 0$ and $b \in \mathbb{R}^d$. Defining $\phi_i(w) = \frac{1}{2}w^T((\mu - \frac{\lambda}{2})I - x_i x_i^T)w - b^T w$ and $\mu > \sigma_1(C) + \frac{\lambda}{2}$ where $\sigma_1(C)$ denotes the largest singular value of C, it also falls into problem (3.1). In this case, $f(w)$ is convex while each $\phi_i(w)$ is probably non-convex.

Distributed machine learning and data mining methods are required to solve the problem (3.1) when the data are distributed over multiple machines. In [43], the authors proposed communication-efficient distributed dual coordinate ascent (CoCoA) for primal-dual distributed optimization. In each iteration, the CoCoA framework allows workers to optimize subproblems independently at first. After that, it calls "Reduce" operation to collect local solution from all workers, and updates global variable and broadcasts the up-to-date global variable to workers in the end. It uses stochastic dual coordinate ascent (SDCA) as the local solver which is one of the most successful methods proposed for solving the problem (3.1)

[36, 91]. In [92], the authors proved that SDCA has linear convergence if the convex function $\phi_i(w)$ is smooth, which is much faster than stochastic gradient descent (SGD). [102, 97] also proposed distributed SDCA and analyzed the tradeoff between computation and communication. [66, 65] accelerated the CoCoA by allowing for more aggressive updates, and proved that CoCoA has linear primal-dual convergence for the smooth convex problem and sublinear convergence for the non-smooth convex problem. However, there are two issues for these primal-dual distributed methods. Firstly, all of them use SDCA as the local solver. SDCA is not applicable when the dual problem is unknown, *e.g.* ϕ_i is non-convex. Therefore, the applications of these primal-dual distributed methods are limited. Secondly, all of these methods assume that the workers have similar computing speed, which is not true in practice. Straggler problem is an unavoidable practical issue in the distributed data mining. Thus, the computing time of CoCoA and distributed SDCA is dependent on the slowest worker. Even if there is only one bad worker, they will work far slower than expectation.

In [89, 90], the authors proposed dual free stochastic dual coordinate ascent (dfSDCA). It was proved to admit similar convergence rate to SDCA while it did not rely on duality at all. However there is no distributed machine learning method using dfSDCA, and its convergence analysis is still unknown yet.

In this chapter, we solve the above two challenging issues in previous primal-dual distributed machine learning methods by proposing novel Distributed Dual Free Stochastic Dual Coordinate Ascent (Dis-dfSDCA). We use dfSDCA as the local solver such that Dis-dfSDCA can be applied to the non-convex problem easily. We alleviate the effect of straggler problem by allowing asynchronous communication between server and workers. As shown in Figure 5, the server does not wait and workers may store the stale global variable in the local. In iteration t, the server receives gradient message v_k from worker k, and sends the up-to-date w^t back to the worker k. Global variables in other workers are stale. For example worker 1 and K store stale global variables w^{t-2} and w^{t-5} respectively. We also analyze the convergence rate of our method and prove that it admits linear convergence rate even if the individual losses (ϕ_i) are non-convex, as long as the sum of losses f is convex. Finally, we conduct simulation on the distributed system with straggler problem. Experimental results verify our theoretical conclusions and show that our method works well in practice.

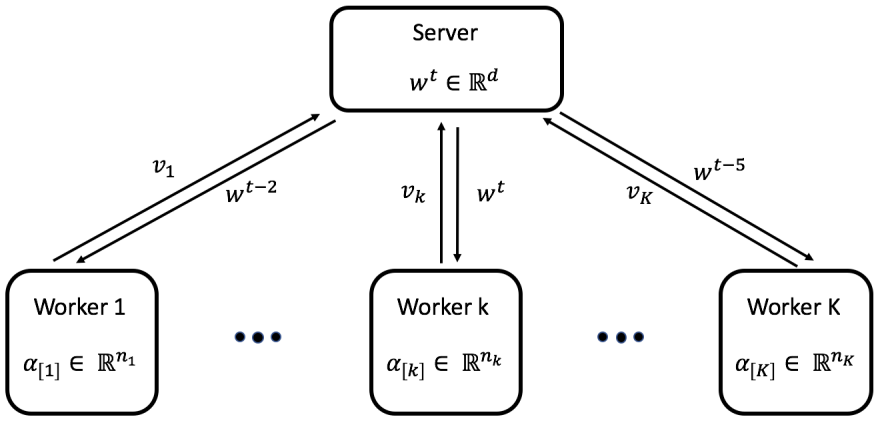


Figure 5: Distributed asynchronous dual free stochastic dual coordinate ascent for parameter server framework.

3.2 Preliminaries

To optimize the primal problem (3.1), we often derive and optimize its dual problem alternatively:

$$\max_{\alpha \in \mathbb{R}^n} D(\alpha) := \max_{\alpha \in \mathbb{R}^n} \frac{1}{n} \sum_{i=1}^{n} -\phi_i^*(-\alpha_i) - \frac{\lambda}{2} \| \frac{1}{\lambda n} A\alpha \|_2^2 \,, \tag{3.2}$$

where ϕ_i^* is the convex conjugate function to ϕ_i, $A = [x_1, x_2, ...x_n] \in \mathbb{R}^{d \times n}$ denotes data matrix and $\alpha \in \mathbb{R}^n$ denotes dual variable. We can use stochastic gradient descent (SGD) to optimize primal problem (3.1), however, there are always two issues: (1) SGD is too aggressive at the beginning of the optimization; (2) it does not have a clear stopping criterion. One of the biggest advantages of optimizing the dual problem is that we can keep tracking the duality gap $G(\alpha)$ to monitor the progress of optimization. Duality gap is defined as: $G(\alpha) = P(w(\alpha)) - D(\alpha)$, where $P(w(\alpha))$ and $D(\alpha)$ denote objective values of primal problem and dual problem respectively. If w^* is the optimal solution of primal problem (3.1) and α^* is the optimal solution of dual problem (3.2), the primal-dual relation always holds that:

$$w^* = w(\alpha^*) = \frac{1}{\lambda n} A\alpha^* \,. \tag{3.3}$$

Algorithm 4 SDCA

1: Initialize α^0 and $w^0 = w(\alpha^0)$;

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

3: Randomly sample i from $\{1, 2, ..., n\}$;

4: Find $\Delta\alpha_i$ to maximize the subproblem (3.4);

5: Update dual variable α through:

$$\alpha^{t+1} \leftarrow \alpha^t + \Delta\alpha_i e_i;$$

6: Update primal variable w through:

$$w^{t+1} \leftarrow w^t + \tfrac{1}{\lambda n}\Delta\alpha_i x_i;$$

7: **end for**

3.2.1 Stochastic Dual Coordinate Ascent

In [92], the authors proposed stochastic dual coordinate ascent (SDCA) to optimize the dual problem (3.2). The pseudocode of SDCA is presented in Algorithm 4. In iteration t, given sample i and other dual variables $\alpha_{j\neq i}$ fixed, we maximize the following subproblem:

$$\max_{\Delta\alpha_i \in \mathbb{R}} -\frac{1}{n}\phi_i^*(-(\alpha_i^t + \Delta\alpha_i)) - \frac{\lambda}{2}\|w^t + \frac{1}{\lambda n}\Delta\alpha_i x_i\|_2^2, \tag{3.4}$$

e_i denotes coordinate vector of size n, where element i is 1 and other elements are 0. In their chapter, the authors proved that SDCA admits linear convergence rate for smooth loss, which is much faster than stochastic gradient descent (SGD). An accelerated SDCA was also proposed in [91]. However, SDCA is not applicable when it is difficult to derive the dual problem, *e.g.* ϕ_i are non-convex.

3.2.2 Dual Free Stochastic Dual Coordinate Ascent

To address the limitation of SDCA, [89] proposes Dual Free Stochastic Coordinate Ascent (dfSDCA) which has similar convergence property to SDCA. The pseudocode of dfSDCA is presented in Algorithm 5. Although we keep vector $\alpha \in \mathbb{R}^n$ in the optimization, the derivation of dual problem is not necessary for dfSDCA. According to the update rule of α and w in the algorithm, the primal-dual relation (3.3) also holds for dfSDCA. The drawback

Algorithm 5 Dual Free SDCA

Initialize dual variable $\alpha^0 = (\alpha_0^0, ..., \alpha_n^0)$ where $\forall i, \alpha_i^0 \in \mathbb{R}^d$, primal variable $w^0 = w(\alpha^0)$;
for $t = 0, 1, 2, \ldots, T-1$ **do**
  Randomly sample i from $\{1, 2, ..., n\}$;
  Compute dual residue κ through:

$$\kappa \leftarrow \nabla\phi_i(w^t) + \alpha_i^t;$$

  Update dual variable α_i through:

$$\alpha_i^{t+1} \leftarrow \alpha_i^t - \eta\lambda n\kappa;$$

  Update primal variable w through:

$$w^{t+1} \leftarrow w^t - \eta\kappa;$$

end for

of dfSDCA is that it is space-consuming to store α, whose space complexity $O(nd)$. We can reduce it to $O(n)$ if $\nabla\phi_i(w)$ can be written as $\nabla\phi_i(x_i^T w)x_i$. In [31], the authors accelerated dfSDCA by using non-uniform sampling strategy in each iteration and proved that it admits faster convergence.

3.3 Distributed Asynchronous Dual Free Stochastic Dual Coordinate Ascent

In this section, we propose Distributed Asynchronous Dual Free Stochastic Coordinate Ascent (Dis-dfSDCA) for distributed optimization. Dis-dfSDCA fits for any parameter server framework, where the star-shape network is used. We assume that there are n samples in the dataset, and they are evenly distributed over K workers. In worker k, there are n_k samples. It is satisfied that $n = \sum_{k=1}^{K} n_k$. Different from sequential dfSDCA, we split the update of dual variable and primal variable into different nodes. The pseudocodes of Dis-dfSDCA for server node and worker nodes are presented in Algorithm 10 and Algorithm 7 respectively.

Algorithm 6 Dis-dfSDCA (Server)

Initialize $w \in \mathbb{R}^d$, η

for $s = 0, 1, ..., S-1$ **do**

for $t = 0, 1, ..., T-1$ **do**

Receive gradient message $v^{s,t} = v_k$ from worker k;

Update global variable $w^{s+1,t+1}$ through:

$$w^{s,t+1} \leftarrow w^{s,t} - \eta v^{s,t};$$

Send $w^{s,t+1}$ back to worker k ;

end for

$w^{s+1,0} = w^{s,T}$

Broadcast the up-to-date global variable $w^{s+1,0}$ to all workers.

end for

3.3.1 Update Global Variable on Server

The up-to-date global variable $w \in \mathbb{R}^d$ is stored and updated on the server. Initially, w is set to be vector zero. At the beginning of each iteration, the server receives gradient message v_k from arbitrary worker k and let $v^t = v_k$. Then it updates the global variable through:

$$w^{s,t+1} = w^{s,t} - \eta v^t. \tag{3.5}$$

Finally, it sends the up-to-date global variable back to the worker k for further computation. Asynchronous method is robust to straggler problem because it allows for updating the global variable when receiving from only one worker. However, if the w in the worker is too stale, it may lead the algorithm to diverge. Therefore, we induce two loops in our algorithm. Server broadcasts the latest global variable w to all workers after every T iterations. In this way, we prevent the problem of divergence and keep the advantage of asynchronous communication at the same time. Algorithm 6 summarizes the pseudocode on the server.

In Algorithm 6, we use the update of vanilla dfSDCA in the server. [90] proposed accelerated dfSDCA by using "Catalyst" algorithm of [62]. It is proved to admit faster convergence

Algorithm 7 Dis-dfSDCA (Worker k)

Initialize $\alpha_{[k]} \in \mathbb{R}^{d \times n_k}$, η, H

repeat

Receive global variable $w^{s,d(t)}$ from server;

Initialize gradient message: $v_k \leftarrow 0$;

Randomly select samples I_t from $\{1, \cdots, n_k\}$ where $|I_t| = H$;

for sample i in I_t **do**

Compute dual residue κ through:

$$\kappa \leftarrow \nabla \phi_i(w^{s,d(t)}) + \alpha_i;$$

Update local dual variable α_i through:

$$\alpha_i \leftarrow \alpha_i - \eta \lambda n \kappa;$$

Update gradient message v_k through:

$$v_k \leftarrow v_k + \kappa;$$

end for

Send gradient message v_k to server;

until Termination

rate by a constant factor. Our Algorithm 6 can also be extended to the accelerated version easily. In this chapter, we only consider the vanilla version and analyze the convergence rate of our algorithm.

3.3.2 Update Local Variable on Worker

In the distributed optimization, workers are responsible for the gradient computation which is the main workload during the optimization. We take arbitrary worker k as an example. Dual variable $\alpha_{[k]} \in \mathbb{R}^{n_k}$ is only stored and updated in the worker k, each α_i is corresponding to sample i. Initially, local variable $\alpha_{[k]}$ is set to be vector zero. After receiving stale global variable $w^{s,d(t)} \in \mathbb{R}^d$ from the server, worker k computes the dual residue κ and updates local variable α_i and gradient message v_k for H iterations. Samples I_t are randomly selected in the local dataset, and we set $|I_t| = H$. In each iteration, worker k selects a sample

i randomly and computes the dual residue κ for coordinate i of the dual variable through the following function:

$$\kappa = \nabla\phi_i(w^{s,d(t)}) + \alpha_i. \tag{3.6}$$

Dual residue can also be viewed as the gradient in Stochastic Gradient Descent. When we obtain optimal dual variable α^* and primal variable w^*, κ should be 0. Therefore, it is satisfied that $\alpha_i^* = -\nabla\phi_i(w^*)$. Then worker k updates local dual variable α_i and gradient message v_k separately through:

$$\alpha_i = \alpha_i - \eta\lambda n\kappa, \quad i \in I_t \tag{3.7}$$

$$v_k = v_k + \kappa. \tag{3.8}$$

Because there is only one α_i in the cluster, it is always up-to-date. After H iterations, the worker k sends gradient message v_k to the server. From the update rule in our algorithm, it is easy to know that the well-known primal-dual relation in the equation (3.3) is always satisfied. The pseudocode of Dis-dfSDCA in worker node k is described in Algorithm 7.

In Algorithm 7, we use vanilla dfSDCA in the worker which samples with uniform distribution. There are also other sampling techniques proposed to accelerate dfSDCA. As per the sampling strategy in [89, 31, 90, 14], there are three options: uniform sampling, importance sampling, and adaptive sampling. In importance sampling strategy [90], it first computes the fixed probability distribution p_i using smoothness parameter of each function ϕ_i, then selects samples following this probability. In adaptive sampling strategy [31], it computes the adaptive probability distribution p_i using dual residue κ for each sample every iteration, then selects samples following this probability. Both of them are proved to admit faster convergence than vanilla dfSDCA with uniform sampling. We only consider the uniform sampling strategy, and analyze its corresponding convergence rate in this chapter. However, other sampling techniques are straightforward to be applied to our distributed method.

3.4 Convergence Analysis

In this section, we provide the theoretical convergence analysis of Dis-dfSDCA. For the case of convex losses ϕ_i, we prove that Dis-dfSDCA admits linear convergence rate. If losses ϕ_i are non-convex, we also prove linear convergence rate as long as the sum-of-non-convex objectives f is convex.

We make the following assumptions for the primal problem (3.1) for further analysis. All of them are common assumptions in the theoretical analysis for the asynchronous stochastic methods.

Assumption 3.4.1 (Lipschitz Constant)**.** *We assume $\nabla\phi_i$ is Lipschitz continuous, and there is Lipschitz constant L such that $\forall x, y \in \mathbb{R}^d$:*

$$\|\nabla\phi_i(x) - \nabla\phi_i(y)\|_2 \leq L\|x - y\|_2. \tag{3.9}$$

We can also know that P is $(L + \lambda)$-smooth:

$$\|\nabla P(x) - \nabla P(y)\|_2 \leq (L + \lambda)\|x - y\|_2. \tag{3.10}$$

Assumption 3.4.2 (Maximum Time Delay)**.** *We assume that the maximum time delay of the global variable in each worker is upper bounded by τ, such that:*

$$d(t) \geq t - \tau. \tag{3.11}$$

τ is relevant to the number of workers K in the system. We can also control τ through inner iteration T in our algorithm.

3.4.1 Convex Case

In this section, we assume that the losses ϕ_i are convex, and prove that our method admits linear convergence.

Assumption 3.4.3 (Convexity). *We assume losses ϕ_i are convex, such that $\forall x, y \in \mathbb{R}^d$:*

$$\phi_i(x) \geq \phi_i(y) + \nabla\phi_i(y)^T(x-y)\,. \tag{3.12}$$

In our algorithm, dual variables $\alpha_{[1]}, ..., \alpha_{[K]}$ are stored in local workers. For worker k, there is no update of $\alpha_{[k]}$ from $d(t)$ to t. Therefore, it is always true that $\alpha_{[k]}^{s,t} = \alpha_{[k]}^{s,d(t)}$. For brevity, we write $v^{s,t}$, $w^{s,t}$ and $\alpha^{s,t}$ as v^t, w^t and α^t. According to our algorithm, we know that:

$$v^t = \sum_{i\in I_t}\left(\nabla\phi_i(w^{d(t)}) + \alpha_i^{d(t)}\right) = \sum_{i\in I_t} v_i^t. \tag{3.13}$$

where $|I_t| = H$ and $\mathbb{E}[v_i^t] = \nabla P(w^{d(t)})$. In our analysis, we also assume that there are no duplicate samples in I_t. To analyze the convergence rate of our method, we need to prove the following Lemma 3.4.1 at first.

Lemma 3.4.1. *Let w^* be the global solution of $P(w)$, and $\alpha_i^* = -\nabla\phi_i(w^*)$. Following the proof in [89], we define A_t and B_t as follows:*

$$A_t = \mathbb{E}\|\alpha_i^t - \alpha_i^*\|^2, \tag{3.14}$$

$$B_t = \mathbb{E}\|w^t - w^*\|^2. \tag{3.15}$$

According to our algorithm, we can prove that A_{t+1} and B_{t+1} are upper bounded:

$$\begin{aligned}\mathbb{E}[A_{t+1} - A_t] \leq\ & -\eta\lambda H\mathbb{E}\|\alpha_i^t - \alpha_i^*\|^2 - 2\eta HL\lambda^2\mathbb{E}\|w^t - w^*\|^2 \\ & +4\eta\lambda HL\left(P(x^t) - P(w^*)\right) - \eta\lambda(1-\eta\lambda n)\mathbb{E}\|v^t\|^2 \\ & +2\lambda\tau HL^2\eta^3\sum_{j=d(t)}^{t-1}\mathbb{E}\|v^j\|^2,\end{aligned} \tag{3.16}$$

$$\begin{aligned}\mathbb{E}[B_{t+1} - B_t] \leq\ & -2\eta\left(P(w^{d(t)}) - P(w^*)\right) + \eta^2\mathbb{E}\|v^t\|^2 \\ & -2\eta\left\langle w^t - w^{d(t)}, \nabla P(x^{d(t)})\right\rangle.\end{aligned} \tag{3.17}$$

Theorem 3.4.1. *Suppose losses ϕ_i are convex and $\nabla\phi_i$ are Lipschitz continuous. Let w^* be the optimal solution to $P(w)$, and $\alpha_i^* = -\nabla\phi_i(w^*)$. Define $C_t = \frac{1}{2\lambda L}A_t + B_t$. We can prove that as long as:*

$$\eta \leq \frac{1}{4HL\tau^2 + \lambda n + 2L}, \tag{3.18}$$

the following inequality holds:

$$\mathbb{E}[C_T] \leq (1 - \eta\lambda H)\mathbb{E}[C_0]. \tag{3.19}$$

Proof of Theorem 3.4.1: Substituting A_{t+1} and B_{t+1} according to Lemma 3.4.1, the following inequality holds that:

$$\begin{aligned}
\mathbb{E}[C_{t+1}] &= \frac{1}{2\lambda L}A_{t+1} + B_{t+1} \\
&\leq (1 - \eta\lambda H)\mathbb{E}[C_t] + 2\tau HL\eta^3 \sum_{j=d(t)}^{t-1} \mathbb{E}\|v^j\|^2 \\
&\quad + \left(\frac{\eta^2\lambda n}{2L} + \eta^2 - \frac{\eta}{2L}\right)\mathbb{E}\|v^t\|^2.
\end{aligned} \tag{3.20}$$

Adding the above inequality from $t = 0$ to $t = T - 1$, we have:

$$\sum_{t=0}^{T-1}\mathbb{E}[C_{t+1}] \leq \sum_{t=0}^{T-1}(1 - \eta\lambda H)\mathbb{E}[C_t] + \left(2H\tau^2\eta^2 + \frac{\eta^2\lambda n}{2L} + \eta^2 - \frac{\eta}{2L}\right)\sum_{t=0}^{T-1}\mathbb{E}\|v^t\|^2, \tag{3.21}$$

where the inequality follows from Assumption 3.4.2 and $\eta L \leq 1$. If $2H\eta^2\tau^2 + \frac{\eta^2\lambda n}{2L} + \eta^2 - \frac{\eta}{2L} \leq 0$, such that:

$$\eta \leq \frac{1}{4HL\tau^2 + \lambda n + 2L}, \tag{3.22}$$

we have the following inequality:

$$\begin{aligned}
\sum_{t=0}^{T-1}\mathbb{E}[C_{t+1}] &\leq \sum_{t=0}^{T-1}(1 - \eta\lambda H)\mathbb{E}[C_t] \\
&\leq \sum_{t=1}^{T-1}\mathbb{E}[C_t] + (1 - \eta\lambda H)C_0.
\end{aligned} \tag{3.23}$$

Because $C_t \geq 0$, then we complete the proof. □

According that $\nabla P(w)$ is Lipschitz continuous, the sub-optimalality $P(w^t) - P(w^*)$ is upper bounded that:

$$\begin{aligned} P(w^t) - P(w^*) &\leq \frac{L+\lambda}{2}\|w^t - w^*\|^2 \\ &\leq \frac{L+\lambda}{2}C_t. \end{aligned} \tag{3.24}$$

Theorem 3.4.2. *We consider the outer iteration s, and write C^t as $C^{s,t}$. According to Algorithm 6, we know $C^{s+1,0} = C^{s,T}$. Following Theorem 3.4.1 and applying (3.19) for S iterations, it is satisfied that:*

$$\mathbb{E}[C_{S,0}] \leq (1 - \eta\lambda H)^S \mathbb{E}[C_{0,0}]. \tag{3.25}$$

In particular, to achieve $\mathbb{E}[P(w^{S,0}) - P(w^*)] \leq \varepsilon$, it suffices to set:

$$\eta = \frac{1}{4HL\tau^2 + \lambda n + 2L}, \tag{3.26}$$

and we have the following result:

$$S \geq O\left(\left(\frac{L}{\lambda}\left(\tau^2 + \frac{1}{H}\right) + \frac{n}{H}\right)\log\left(\frac{1}{\varepsilon}\right)\right). \tag{3.27}$$

From Theorem 3.4.1 and 3.4.2, we know that our Dis-dfSDCA admits linear convergence if losses ϕ_i are convex. According to Theorem 3.4.2, we observe that τ affects the speed of our convergence, if $\tau \to \infty$, it may lead our algorithm to diverge. Therefore, it is important to keep τ within a reasonable bound. In our algorithm, τ is relevant to the number of workers and less than T. When we let $H = 1$ and $\tau = 0$, S is relevant to $O(\frac{L}{\lambda} + n)$. It is compatible with the convergence analysis of sequential dfSDCA in [89].

3.4.2 Non-convex Case

In this section, we assume that the losses ϕ_i are non-convex, while the sum-of-non-convex objectives f is convex. We also prove that Dis-dfSDCA admits linear convergence rate for this case. Firstly, we get the following Lemma 3.4.2.

Lemma 3.4.2. *Let w^* be optimal solution to $P(w)$, and $\alpha_i^* = -\nabla\phi_i(w^*)$. Following the definition of A_t and B_t in Lemma 3.4.1, we prove that A_{t+1} and B_{t+1} are upper bounded:*

$$\begin{aligned}
\mathbb{E}[A_{t+1} - A_t] &\leq -\eta\lambda H\mathbb{E}\|\alpha_i^t - \alpha_i^*\|^2 + 2\eta\lambda HL^2\mathbb{E}\|w^t - w^*\|^2 \\
&\quad -\eta\lambda(1-\eta\lambda n)\mathbb{E}\|v^t\|^2 + 2\lambda\tau HL^2\eta^3\sum_{j=d(t)}^{t-1}\mathbb{E}\|v^j\|^2, \qquad (3.28)\\
\mathbb{E}[B_{t+1} - B_t] &\leq -\frac{3\eta\lambda H}{4}\mathbb{E}\|w^t - w^*\|^2 + \eta^2\mathbb{E}\|v^t\|^2 \\
&\quad +\frac{2H\tau H^2(L+\lambda)^2\eta^3}{\lambda}\sum_{j=d(t)}^{t-1}\mathbb{E}\|v^j\|^2. \qquad (3.29)
\end{aligned}$$

Theorem 3.4.3. *Suppose f is convex and $\nabla\phi_i$ is Lipschitz continuous. Let w^* be the optimal solution to $P(w)$, and $\alpha_i^* = -\nabla\phi_i(w^*)$. Define $C_t = \frac{1}{4L^2}A_t + B_t$. We can prove that as long as:*

$$\eta \leq \frac{\lambda^2}{2HL\tau^2\lambda^2 + 8HL\tau^2(L+\lambda)^2 + 4\lambda L^2 + n\lambda^3}, \qquad (3.30)$$

the following inequality holds:

$$\mathbb{E}[C_T] \leq (1-\eta\lambda H)\mathbb{E}[C_0], \qquad (3.31)$$

Proof of Theorem 3.4.3: Substituting A_{t+1} and B_{t+1} according to Lemma 3.4.2, the following inequality holds that:

$$\begin{aligned}
\mathbb{E}[C_{t+1}] &= \frac{1}{4L^2}A_{t+1} + B_{t+1} \\
&\leq (1-\eta\lambda H)\mathbb{E}[C_t] + \left(\frac{\lambda H\tau\eta^3}{2} + \frac{2H\tau(L+\lambda)^2\eta^3}{\lambda}\right)\sum_{j=d(t)}^{t-1}\mathbb{E}\|v^j\|^2 \\
&\quad + \left(\eta^2 + \frac{n\eta^2\lambda^2}{4L^2} - \frac{\eta\lambda}{4L^2}\right)\mathbb{E}\|v^t\|^2. \qquad (3.32)
\end{aligned}$$

Adding the above inequality from $t = 0$ to $t = T - 1$, we have:

$$\sum_{t=0}^{T-1} \mathbb{E}[C_{t+1}] \leq \sum_{t=0}^{T-1} (1 - \eta\lambda H)\mathbb{E}[C_t] + \left(\eta^2 + \frac{n\eta^2\lambda^2}{4L^2} + \frac{\lambda H\tau^2\eta^2}{2L} + \frac{2H\tau^2(L+\lambda)^2\eta^2}{\lambda L} - \frac{\eta\lambda}{4L^2}\right) \sum_{t=0}^{T-1} \mathbb{E}\|v^t\|^2, \tag{3.33}$$

where the inequality follows from Assumption 3.4.2 and $\eta L \leq 1$. If $\eta^2 + \frac{n\eta^2\lambda^2}{4L^2} + \frac{\lambda H\tau^2\eta^2}{2L} + \frac{2H\tau^2(L+\lambda)^2\eta^2}{\lambda L} - \frac{\eta\lambda}{4L^2} \leq 0$, such that:

$$\eta \leq \frac{\lambda^2}{2HL\tau^2\lambda^2 + 8HL\tau^2(L+\lambda)^2 + 4\lambda L^2 + n\lambda^3}, \tag{3.34}$$

we have the following inequality:

$$\begin{aligned} \sum_{t=0}^{T-1} \mathbb{E}[C_{t+1}] &\leq \sum_{t=0}^{T-1} (1 - \eta\lambda H)\mathbb{E}[C_t] \\ &\leq \sum_{t=1}^{T-1} \mathbb{E}[C_t] + (1 - \eta\lambda H)C_0. \end{aligned} \tag{3.35}$$

Because $C_t \geq 0$, then we complete the proof that $\mathbb{E}[C_T] \leq (1 - \eta\lambda H)\mathbb{E}[C_0]$.

□

Theorem 3.4.4. *We consider the outer iteration s, and write C^t as $C^{s,t}$. According to Algorithm 6, we know $C^{s+1,0} = C^{s,T}$. Following Theorem 3.4.3 and applying (3.31) for S iterations, it is satisfied that:*

$$\mathbb{E}[C_{S,0}] \leq (1 - \eta\lambda H)^S \mathbb{E}[C_{0,0}]. \tag{3.36}$$

To achieve $\mathbb{E}[P(w^{S,0}) - P(w^*)] \leq \varepsilon$, it suffices to set $\eta = \frac{\lambda^2}{2HL\tau^2\lambda^2 + 8HL\tau^2(L+\lambda)^2 + 4\lambda L^2 + n\lambda^3}$ and

$$S \geq O\left(\left(\frac{(\tau^2 + 1/H)\, L^2}{\lambda^2} + \frac{\tau^2 L^3}{\lambda^3} + \frac{n}{H}\right) \log\left(\frac{1}{\varepsilon}\right)\right). \tag{3.37}$$

From Theorems 3.4.3 and 3.4.4, we know that our Dis-dfSDCA admits linear convergence even if losses ϕ_i are non-convex, as long as the sum-of-non-convex objectives is convex. Comparing Theorems 3.4.2 with 3.4.4, we can observe that our method needs more iterations to converge to the similar accuracy when ϕ_i are non-convex. It is reasonable because non-convex problem is known to be harder to be optimized than convex problem. When we let $H = 1$ and $\tau = 0$, S is relevant to $O(\frac{L^2}{\lambda^2} + n)$. It is also compatible with the convergence analysis of sequential dfSDCA in [89].

3.5 Experimental Results

In this section, we conduct two simulated experiments on the distributed system with straggler problem. There are mainly three goals, firstly, we want to verify that our DisdfSDCA has linear convergence rate for the convex and smooth problem; secondly, we would like to make sure that our method has better speedup property than other primal-dual methods; thirdly, we would like to show that our method is also fit for non-convex losses.

Our algorithm is implemented using C++, and the point-to-point communication between worker and server is handled by openMPI [22]. We use Armadillo library [86] for efficient matrix computation. Experiments are performed on Amazon Web Services, and each node is a t2.medium instance which has two virtual CPUs. In our distributed system, we simulate the straggler problem by forcing one selected worker node to the delaying state for m times as long as the normal computing time of other normal workers with probability p. In our experiments, we set $p = 0.2$ and m is selected from $[0, 10]$ randomly. In practice, all nodes have a tiny possibility of being delayed. The setting in our experiments is to verify that our algorithm is robust to straggler problem, even in the extreme situation.

3.5.1 Convex Case

In our experiment, we optimize quadratic loss with ℓ_2 regularization term to solve binary classification problem:

$$\min_{w\in\mathbb{R}^d} \frac{1}{n}\sum_{i=1}^{n} \frac{1}{2}(x_i^T w - y_i)^2 + \frac{\lambda}{2}\|w\|^2, \tag{3.38}$$

where $\lambda = 0.1$. Datasets in our experiments are from LIBSVM [12]. Table 2 shows brief details of each dataset. In this problem, because $\nabla\phi_i(w)$ can be written as $\nabla\phi_i(x_i^T w)$, we just need to store $\hat{\alpha} \in \mathbb{R}^n$, and recover $\alpha \in \mathbb{R}^{d\times n}$ through $a_i = x_i\hat{\alpha}_i$. Therefore the space complexity is $O(n)$.

We compare our method with CoCoA+ [66], which is the state-of-the-art distributed primal-dual optimization framework. We reimplement CoCoA+ framework using C++, and use SDCA as the local solver. Learning rate η in our method is selected from $\eta = \{1, 0.1, 0.001, 0.0001\}$.

Table 2: Experimental datasets from LIBSVM.

Dataset	# of samples	Dimension	Sparsity
IJCNN1	49,990	22	41 %
COVTYPE	581,012	54	22 %
RCV1	677,399	47,236	0.16%

3.5.1.1 Convergence of Duality Gap We compare the duality gap convergence of compared methods in terms of time and epoch number respectively, where duality gap is well defined in [92]. Experimental results are presented in Figure 6. We distribute IJCNN1 dataset over 4 workers. Figures 6 show the duality gap convergence in terms of time and epoch on IJCNN1 dataset. From the second figure, it is easy to know that Dis-dfSDCA and CoCoA+ have similar convergence rate. Since CoCoA+ has linear convergence if the problem is convex and smooth, it is verified that Dis-dfSDCA has linear convergence rate as well. In the experiment, we evaluate Dis-dfSDCA when we set different amount of local computations, $H = 10^2$ and $H = 10^3$. Results show that our method is faster than CoCoA+ method in both two cases. The reason is that CoCoA+ is affected by the straggler problem in the distributed system. We also optimize problem (3.38) with COVTYPE dataset using 8 workers, and RCV1 dataset using 16 workers. We can draw the similar conclusion from the results of other two datasets.

3.5.1.2 Speedup In this section, we evaluate the scaling up ability of compared methods. The first row of Figure 7 presents the speedup of compared methods on IJCNN1 and COVTYPE datasets. Speedup is defined as follows:

$$\text{Time speedup } = \frac{\text{Running time for serial computation}}{\text{Running time of using } K \text{ workers}}. \tag{3.39}$$

Figure in the second row shows the convergence of duality gap on RCV1 on multiple machines. It is obvious that Dis-dfSDCA always converges faster than CoCoA+ when they

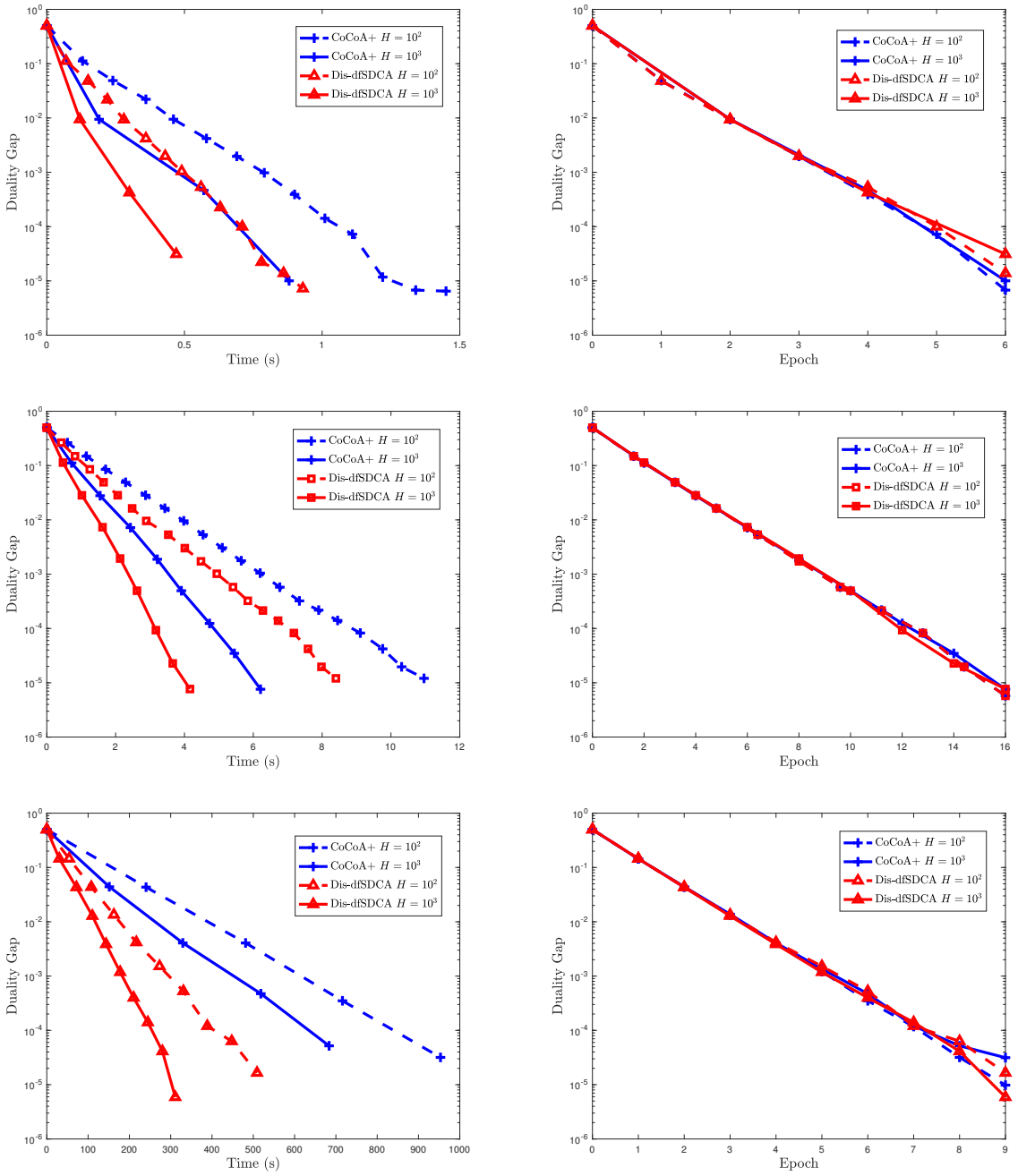


Figure 6: Convergence of duality gap of compared methods in terms of time and epoch for IJCNN1, COVTYPE, RCV1 respectively.

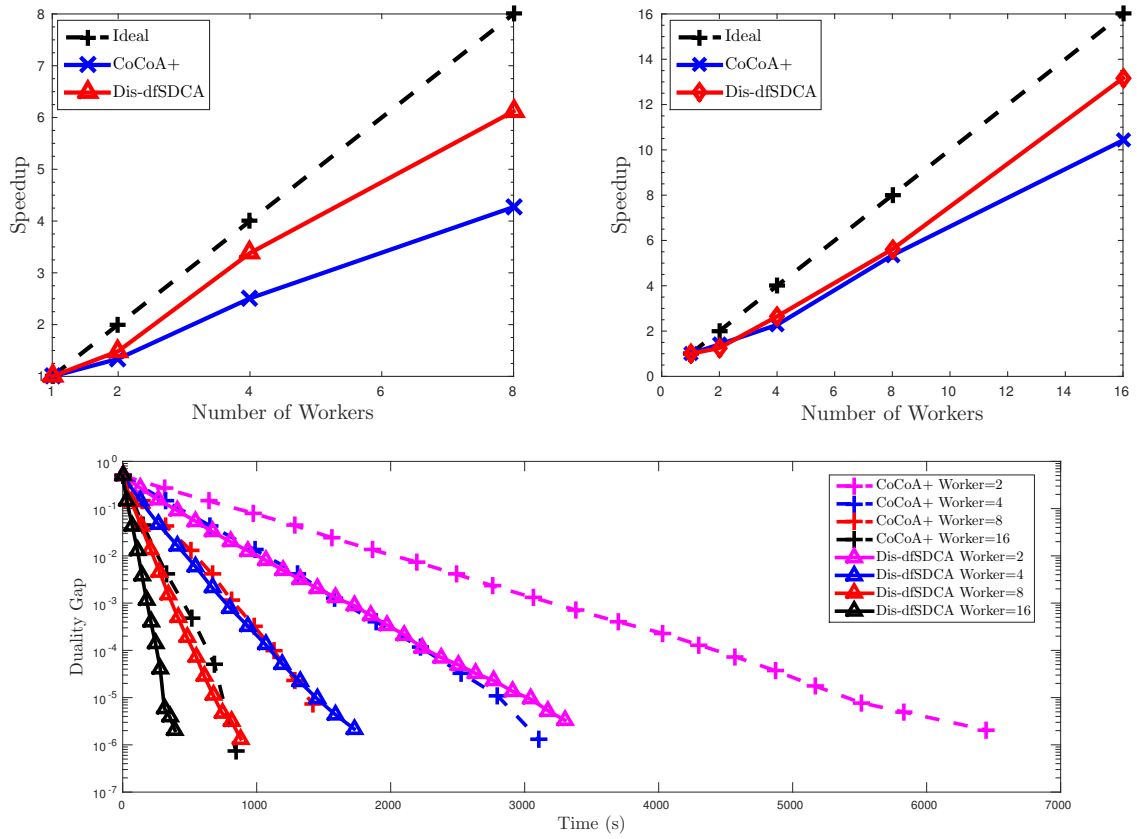


Figure 7: Time speedup in terms of the number of workers. Row 1 left: IJCNN1; Row 1 right: COVTYPE; Row 2: RCV1.

have the same number of workers. Experimental results verify that Dis-dfSDCA has better speedup property than CoCoA+ when there is straggler problem.

3.5.2 Non-convex Case

In this experiment, we optimize the following convex objective, which is an essential step for principal component analysis in [23]:

$$\min_{w\in\mathbb{R}^d} \frac{1}{n}\sum_{i=1}^{n} \frac{1}{2} w^T \left((\mu-\lambda) - x_i x_i^T\right) w - b^T w + \frac{\lambda}{2}\|w\|^2. \tag{3.40}$$

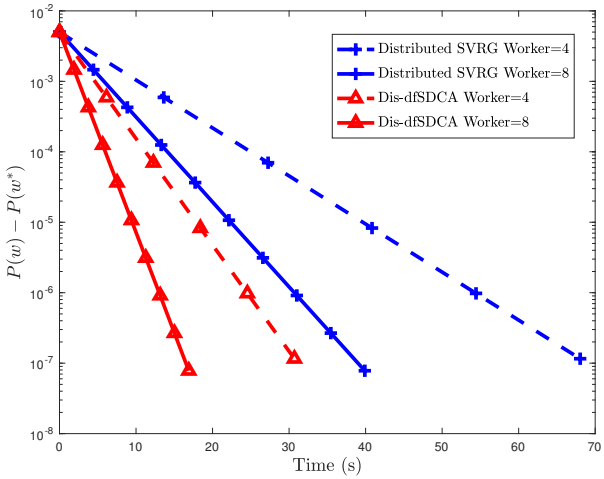


Figure 8: Suboptimum convergence of compared methods in terms of time.

We conduct the experiment on synthetic data and generate $n = 500,000$ random vectors $\{x_1, ..., x_{500,000}\} \in \mathbb{R}^{500}$ which are mean subtracted and normalized to have Euclidean norm 1. $C = \frac{1}{n}\sum_{i=1}^{n} x_i x_i^T$ denotes covariance matrix, $b \in \mathbb{R}^d$ denotes a random vector and we let $\mu = 100$, $\lambda = 10^{-4}$ in the experiment. Because each ϕ_i is probably non-convex, CoCoA is not able to solve this problem. In this experiment, we compare with Distributed asynchronous SVRG [39].

Figure 8 shows the suboptimum convergence $P(w) - P(w^*))$ regarding time. w^* denotes the optimal solution to problem (3.40) , and it is obtained by running Dis-dfSDCA until convergence. In Figure 8, it is obvious that Dis-dfSDCA runs faster than Distributed SVRG when there are 4 workers. We can observe the similar phenomenon when there are 8 workers. This observation is reasonable because Distributed SVRG needs to compute two gradients in each inner iteration and full gradient in each outer iteration. Dis-dfSDCA is faster because it only needs to compute one gradient in each iteration. However, Dis-dfSDCA needs $O(nd)$ space for storing α , because $\nabla\phi_i(w)$ cannot be written as $\nabla\phi_i(x_i^T w)x_i$ in this problem.

4.0 Large Batch Training Does Not Need Warmup

4.1 Motivation

Deep learning has made significant breakthroughs in many fields, such as computer vision [30, 29, 54, 83], nature language processing [19, 34, 100], and reinforcement learning [70, 94]. Recent studies show that better performance can usually be achieved by training a larger neural network with a bigger dataset [67, 79]. Nonetheless, it is time-consuming to train deep neural networks, which limits the efficiency of deep learning research. For example, training ResNet50 on ImageNet with batch size 256 needs to take about 29 hours to obtain 75.3% Top-1 accuracy on 8 Tesla P100 GPUs [30]. Thus, it is a critical topic to reduce the training time for the development of deep learning. Data parallelism is the most popular method to speed up the training process, where the large-batch data is split across multiple devices [16, 52, 101]. However, the large-batch neural network training using conventional optimization techniques usually leads to bad generalization errors [35, 48].

Many empirical training techniques have been proposed for large-batch deep learning optimization. [28] proposed to adjust the learning rate through linear learning rate scaling and gradual warmup. By using these two techniques, they successfully trained ResNet50 with a batch size of 8192 on 256 GPUs in one hour with no loss of accuracy. Most of the theoretical analysis about linear learning rate scaling consider stochastic gradient descent only [59, 60]. However, the theoretical analysis for the momentum method or Nesterov's Accelerated Gradient [74] is still unknown. Finding that the ratios of weight's ℓ_2-norm to gradient's ℓ_2-norm vary greatly among layers, [106] proposed the state-of-the-art large-batch optimizer Layer-wise Adaptive Rate Scaling (LARS) and scaled the batch size to 16384 for training ResNet50 on ImageNet. However, LARS still requires warmup in early epochs of training and may diverge if it is not tuned properly.

Above three techniques (linear learning rate scaling, gradual warmup, and LARS) are demonstrated to be very effective and have been applied in many related works reducing the training time of deep neural networks [3, 44, 68, 105, 107]. In spite of the effectiveness

of above training techniques, theoretical motivations behind these techniques are still open problems: (I) Why we need to increase the learning rate linearly as batch size scales up? (II) Why we use gradual warm at early epochs, does there exist an optimal warmup technique with no need to tune hyper-parameters? (III) Why we need to adjust the learning rate layer-wisely?

In this chapter, we target to remove the warmup technique for large-batch training and bridge the gap between large-batch deep learning optimization heuristics and theoretical underpins. We summarize our main contributions as follows:

1. We propose a novel Complete Layer-wise Adaptive Rate Scaling (CLARS) algorithm for large-batch deep learning optimization. Then, we introduce a new fine-grained analysis for gradient-based methods and prove that the proposed method is guaranteed to converge for non-convex problems.
2. We bridge the gap between heuristics and theoretical analysis for three large-batch deep learning optimization techniques, including layer-wise adaptive rate scaling, linear learning rate scaling, and gradual warmup.
3. Extensive experimental results demonstrate that CLARS outperforms gradual warmup by a large margin and defeats the convergence of the state-of-the-art large-batch optimizer in training advanced deep neural networks (ResNet, DenseNet, MobileNet) on ImageNet dataset.

4.2 Preliminaries

Gradient-Based Methods: The loss function of a neural network is minimizing the average loss over a dataset of n samples:

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

where d denotes the dimension of the neural network. Normally, first-order gradient methods are used to optimizer deep learning models. Momentum-based methods have been widely used in deep learning optimization, especially computer vision, and obtain state-of-the-art

results [30, 37]. According to [74], mini-batch Nesterov Accelerated Gradient (mNAG) optimizes the problem (4.1) as follows:

$$\begin{aligned} v_{t+1} &= w_t - \gamma \frac{1}{B} \sum_{i \in I_t} \nabla f_i(w_t), \\ w_{t+1} &= v_{t+1} + \beta(v_{t+1} - v_t), \end{aligned} \tag{4.2}$$

where I_t is the mini-batch samples with $|I_t| = B$, γ is the learning rate, $\beta \in [0, 1)$ is the momentum constant and v is the momentum vector. When $\beta = 0$, Eq. (4.2) represents the procedures of mini-batch Gradient Descent (mGD). Learning rate γ is scaled up linearly when batch size B is large [28]. However, using a learning rate γ for all layers may lead to performance degradation.

Layer-Wise Learning Rate Scaling: To train neural networks with large batch size, [106] proposed Layer-Wise Adaptive Rate Scaling (LARS). Suppose a neural network has K layers, we can rewrite $w = [(w)_1, (w)_2, ..., (w)_K]$ with $(w)_k \in \mathbb{R}^{d_k}$ and $d = \sum_{k=1}^{K} d_k$. The learning rate at layer k is updated as follows:

$$\gamma_k \quad = \quad \gamma_{scale} \times \eta \times \frac{\|(w_t)_k\|_2}{\left\| \frac{1}{B} \sum_{i \in I_t} \nabla_k f_i(w_t) \right\|_2}, \tag{4.3}$$

where $\gamma_{scale} = \gamma_{base} \times \frac{B}{B_{base}}$ and $\eta = 0.001$ in [106]. γ_{base} and B_{base} depends on model and dataset. For example, we set $\gamma_{base} = 0.1$ and $B_{base} = 128$ to train ResNet on CIFAR10. Although LARS works well in practice, there is little theoretical understanding about it and it converges slowly or even diverges in the beginning of training if warmup [28] is not used .

Conventional Analysis: [10, 25, 103] proved the convergence of mGD or mNAG for non-convex problems through following two Assumptions:

Assumption 4.2.1. (Lipschitz Continuous Gradient) *The gradient of f is Lipschitz continuous with constant L_g. For any $w, v \in \mathbb{R}^d$, it is satisfied that:*

$$\|\nabla f(w) - \nabla f(w + v)\|_2 \leq L_g \|v\|_2. \tag{4.4}$$

Assumption 4.2.2. (Bounded Variance) *There exist constants $M_g > 0$ and $M_C > 0$, for any $w \in \mathbb{R}^d$, it is satisfied that:*

$$\mathbb{E} \|\nabla f_i(w) - \nabla f(w)\|_2^2 \leq M_g \mathbb{E} \|\nabla f(w)\|_2^2 + M_C. \tag{4.5}$$

Theorem 4.2.1 ([103]). *Under Assumptions 4.2.1 and 4.2.2, let $f_{\inf}$ denote the minimum value of problem $f(w)$ and $M_g = 0$. As long as $\gamma \leq \frac{1-\beta}{2L}$, $\min_{t=0,1,...,T-1} \|\nabla f(w_t)\|_2^2$ is guaranteed to converge at the rate of $O\left(\frac{1}{T\gamma} + L_g \gamma M_C\right)$.*

From Theorem 4.2.1, it is natural to know that the value of γ should be lowered because of the term $O(L_g \gamma M_C)$, which is consistent with the learning rate decay practically. However, there are two weaknesses of the current convergence result: (I) It cannot explain why layer-wise learning rate in [106] is useful when there is one γ for all layers. (II) Theoretical result doesnot show that warmup is required in the early stage of training.

4.3 Complete Layer-Wise Adaptive Rate Scaling

In this section, we propose a novel Complete Layer-wise Adaptive Rate Scaling (CLARS) algorithm for large-batch deep learning optimization and a new fine-grained convergence analysis of gradient-based methods for non-convex problems.

4.3.1 Complete Layer-Wise Adaptive Rate Scaling

Define $U \in \mathbb{R}^{d \times d}$ as a permutation matrix where every row and column contains precisely a single 1 with 0s everywhere else. Let $U = [U_1, U_2, ..., U_K]$ and U_k corresponds to the parameters of layer k, the relation between w and w_k is $w = \sum_{k=1}^{K} U_k w_k$. Let $\nabla_k f_i(w_t)$ denote the stochastic gradient with respect to the parameters at layer k and γ_k denote its learning rate. Thus, Eq. (4.2) of mNAG with batch I_t can be rewritten as:

$$\begin{cases} v_{t+1} &= w_t - \sum_{k=1}^{K} \gamma_k U_k \left(\frac{1}{B} \sum_{i \in I_t} \nabla_k f_i(w_t) \right) \\ w_{t+1} &= v_{t+1} + \beta(v_{t+1} - v_t) \end{cases}. \tag{4.6}$$

At each iteration, the learning rate γ_k at layer k is updated using Complete Layer-wise Adaptive Rate Scaling (CLARS) as follows:

$$\gamma_k = \gamma_{scale} \times \eta \times \frac{\|(w_t)_k\|_2}{\frac{1}{B} \sum_{i \in I_t} \|\nabla_k f_i(w_t)\|_2}, \tag{4.7}$$

Algorithm 8 Complete Layer-Wise Adaptive Rate Scaling

Require: γ_{scale}: Maximum learning rate

Require: β: Momentum parameter

Require: $\eta = 0.01$

1: **for** $t = 0, 1, 2, \cdots, T-1$ **do**

2: Sample large-batch I_t randomly with batch size B;

3: Compute large-batch gradient $\frac{1}{B}\sum_{i\in I_t}\nabla f_i(w_t)$;

4: Compute the average of gradient norm for K layers $\frac{1}{B}\sum_{i\in I_t}\|\nabla_k \nabla f_i(w_t)\|_2^2$;

5: Update layer-wise learning rate γ_k following Eq. (4.7);

6: Update the model w_t and momentum term v_t following Eq. (4.6);

7: **end for**

8: Output w_T as the final result.

where $\gamma_{scale} = \gamma_{base} \times \frac{B}{B_{base}}$ and η is constant. To obtain a clear understanding of Eq. (4.7), we rewrite it as:

$$\gamma_k = \gamma_{scale} \times \eta \times \frac{\|(w_t)_k\|_2}{\left\|\frac{1}{B}\sum_{i\in I_t}\nabla_k f_i(w_t)\right\|_2} \times \frac{\left\|\frac{1}{B}\sum_{i\in I_t}\nabla_k f_i(w_t)\right\|_2}{\frac{1}{B}\sum_{i\in I_t}\|\nabla_k f_i(w_t)\|_2}. \tag{4.8}$$

It is equal to multiplying the LARS in Eq. (4.3) with a new term $\frac{\left\|\frac{1}{B}\sum_{i\in I_t}\nabla_k f_i(w_t)\right\|_2}{\frac{1}{B}\sum_{i\in I_t}\|\nabla_k f_i(w_t)\|_2}$, which plays a critical role in removing the warmup. The proposed CLARS method is briefly summarized in Algorithm 8.

In the following section, we will show that CLARS is supported theoretically and the learning rate at layer k is normalized with respect to its corresponding Lipschitz constant and gradient variance. In the experiments, we will also demonstrate that the proposed method can complete large-batch ImageNet training with no warmup for the first time and accelerate the convergence.

4.3.2 Fine-Grained Micro-Steps and Assumptions

In this section, we propose a new fine-grained method for the convergence analysis of gradient-based methods. Based on the fine-grained analysis, we prove the convergence rate

of mini-batch Gradient Descent (mGD) and mini-batch Nesterov's Accelerated Gradient (mNAG) for deep learning problems. More insights are obtained by analyzing their convergence properties.

Each step of mNAG in Eq. (4.6) can be regarded as the result of updating v, w for K micro-steps, where the gradient at each micro-step is $\frac{1}{B}\sum_{i\in I_t}\nabla_k f_i(w_t)$. At micro-step $t{:}s$, we have layer index $k(s) = s \pmod K + 1$. For example, when $s = 0$, we are updating the parameters of layer $k(0) = 1$. Defining $w_{t:0} = w_t$, $w_{t:K} = w_{t+1}$, we can obtain Eq. (4.6) after applying following equations from $s = 0$ to $s = K - 1$:

$$\begin{cases} v_{t:s+1} &= w_{t:s} - \frac{\gamma_k}{B}\sum\limits_{i\in I_t} U_k\nabla_k f_i(w_t) \\ w_{t:s+1} &= v_{t:s+1} + \beta(v_{t:s+1} - v_{t:s}) \end{cases}. \tag{4.9}$$

Following the idea of block-wise Lipschitz continuous assumption in [7] and regarding layers as blocks, we suppose that two layer-wise assumptions are satisfied for any K-layer neural network throughout this chapter, .

Assumption 4.3.1 (Layer-Wise Lipschitz Continuous Gradient)**.** *Assume that the gradient of f is layer-wise Lipschitz continuous and the Lipschitz constant corresponding to layer k is L_k for any layer $k \in \{1, 2, ..., K\}$. For any $w \in \mathbb{R}^d$ and $v = [v_1, v_2, ..., v_K] \in \mathbb{R}^d$, the following inequality is satisfied that for any $k \in \{1, 2, ..., K\}$:*

$$\|\nabla_k f(w) - \nabla_k f(w + U_k v_k)\|_2 \;\leq\; L_k\|v_k\|_2. \tag{4.10}$$

Lipschitz constants L_k of different layers are not equal and can be affected by multiple factors, for example, position (top or bottom) or layer type (CNN or FCN). [113] estimated Lipschitz constants empirically and verified that Lipschitz constants of gradients at different layers vary a lot. L_k represents the property at layer k and plays an essential role in tuning learning rates. In addition, we also think the "global" Lipschitz continuous assumption in Assumption 4.2.1 is satisfied and $L_g \geq L_k$.

Assumption 4.3.2 (Layer-Wise Bounded Variance)**.** *Assume that the variance of stochastic gradient with respect to the parameters of layer k is upper bounded. For any $k \in \{1, 2, ..., K\}$ and $w \in \mathbb{R}^d$, there exists $M_k > 0$ and $M > 0$ so that:*

$$\mathbb{E} \left\| \nabla_k f_i(w) - \nabla_k f(w) \right\|_2^2 \leq M_k \mathbb{E} \|\nabla_k f(w)\|_2^2 + M. \tag{4.11}$$

Let $M_k \leq M_g$ for any k, it is straightforward to get the upper bound of the variance of gradient $\nabla f_i(w)$ as $\mathbb{E} \left\| \nabla f_i(w) - \nabla f(w) \right\|_2^2 \leq M_g \mathbb{E} \|\nabla f(w)\|_2^2 + KM$. It is obvious that the value of $M_C = KM$ in Assumption 4.2.2 is dependent on the neural networks depth.

Difficulties of Convergence Analysis: There are two major difficulties in proving the convergence rate using the proposed fine-grained micro-steps. (I) Micro-step induces stale gradient in the analysis. At each micro-step $t{:}s$ in Eq. (4.9), gradient is computed using the stale model w_t, rather than the latest model $w_{t:s}$. (II) K Lipschitz constants for K layers are considered separately and simultaneously, which is much more complicated than just considering L_g for the whole model.

4.3.3 Convergence Guarantees of Two Gradient-Based Methods

Based on the proposed fine-grained analysis, we prove that both of mini-batch Gradient Descent (mGD) and mini-batch Nesterov's Accelerated Gradient (mNAG) admit sub-linear convergence guarantee $O\left(\frac{1}{\sqrt{T}}\right)$ for non-convex problems. Finally, we obtain some new insights about the gradient-based methods by taking mNAG as an example. At first, we let $\beta = 0$ in Eq. (4.6) and Eq. (4.9), and analyze the convergence of mGD method.

Lemma 4.3.1. *Under Assumptions 4.3.1 and 4.3.2, after applying Eq. (4.9) with $\beta = 0$ for K micro-steps from $s = 0$ to $s = K - 1$, we have the upper bound of loss $\mathbb{E}[f(w_{t+1})]$ as follows:*

$$\begin{aligned} \mathbb{E}\left[f(w_{t+1})\right] \;\leq\; & \mathbb{E}\left[f(w_t)\right] - \sum_{k=1}^{K} \frac{\gamma_k}{2} \Bigg(1 - L_k \gamma_k - \frac{L_k \gamma_k M_k}{B} - \frac{L_g^2 \gamma_k M_k \sum_{k=1}^{K} \gamma_k}{KB} \\ & - \frac{L_g^2 \gamma_k \sum_{k=1}^{K} \gamma_k}{K} \Bigg) \mathbb{E} \left\| \nabla_k f(w_t) \right\|_2^2 + \sum_{k=1}^{K} \frac{L_k \gamma_k^2 M}{2B} + \frac{\sum_{k=1}^{K} \gamma_k L_g^2 M}{2KB} \sum_{k=1}^{K} \gamma_k^2. \end{aligned} \tag{4.12}$$

Proof of Lemma 4.3.1: Suppose K layers are updated sequentially from $s = 0$ to $K-1$, and we have $w_t = w_{t:0}$ and $w_{t+1} = w_{t:K}$. At micro-step $t{:}s$, we set $k(s) = s \pmod K + 1$. According to Assumption 4.3.1, we have:

$$\begin{aligned}
\mathbb{E}\left[f(w_{t:s+1})\right] &\leq \mathbb{E}\left[f(w_{t:s})\right] - \mathbb{E}\left\langle \nabla_{k(s)} f(w_{t:s}), \gamma_{k(s)} \nabla_{k(s)} f(w_t)\right\rangle \\
&\quad + \frac{L_{k(s)}\gamma_{k(s)}^2}{2} \mathbb{E}\left\| \frac{1}{B} \sum_{i \in I_t} \nabla_{k(s)} f_i(w_t) \right\|_2^2 \\
&= \mathbb{E}\left[f(w_{t:s})\right] + \frac{L_{k(s)}\gamma_{k(s)}^2}{2} \underbrace{\mathbb{E}\left\| \frac{1}{B} \sum_{i \in I_t} \nabla_{k(s)} f_i(w_t) \right\|_2^2}_{C_1} - \frac{\gamma_{k(s)}}{2} \Bigg(\mathbb{E}\left\| \nabla_{k(s)} f(w_{t:s}) \right\|_2^2 \\
&\quad + \mathbb{E}\left\| \nabla_{k(s)} f(w_t) \right\|_2^2 - \underbrace{\mathbb{E}\left\| \nabla_{k(s)} f(w_{t:s}) - \nabla_{k(s)} f(w_t) \right\|_2^2}_{C_2} \Bigg). \qquad (4.13)
\end{aligned}$$

In the following context, we will prove that C_1 and C_2 are upper bounded. At first, we can get the upper bound of C_1 as follows:

$$\begin{aligned}
C_1 &= \mathbb{E}\left\| \frac{1}{B} \sum_{i \in I_t} \left(\nabla_{k(s)} f_i(w_t) - \nabla_{k(s)} f(w_t) + \nabla_{k(s)} f(w_t) \right) \right\|_2^2 \\
&= \frac{1}{B^2} \mathbb{E}\left\| \sum_{i \in I_t} \left(\nabla_{k(s)} f_i(w_t) - \nabla_{k(s)} f(w_t) \right) \right\|_2^2 + \mathbb{E}\left\| \nabla_{k(s)} f(w_t) \right\|_2^2 \\
&\leq \frac{1}{B^2} \sum_{i \in I_t} \mathbb{E}\left\| \nabla_{k(s)} f_i(w_t) - \nabla_{k(s)} f(w_t) \right\|_2^2 + \mathbb{E}\left\| \nabla_{k(s)} f(w_t) \right\|_2^2 \\
&\leq \left(1 + \frac{M_{k(s)}}{B} \right) \mathbb{E}\left\| \nabla_{k(s)} f(w_t) \right\|_2^2 + \frac{M}{B}, \qquad (4.14)
\end{aligned}$$

where the second equality follows from $\mathbb{E}\left\langle \nabla_{k(s)} f_i(w_t) - \nabla_{k(s)} f(w_t), \nabla_{k(s)} f(w_t) \right\rangle = 0$ and the first inequality follows from $\mathbb{E}\left\| \sum_{i=1}^n \xi_i \right\|_2^2 \leq \sum_{i=1}^n \mathbb{E}\left\| \xi_i \right\|_2^2$ if $\mathbb{E}[\xi_i] = 0$ and the second inequality

follows from Assumption 4.3.2. Following "global" Lipschitz continuous in Assumption 4.2.1, we can bound C_2 as follows:

$$
\begin{aligned}
C_2 &\leq \frac{L_g^2}{K}\mathbb{E}\left\|w_{t:s}-w_t\right\|_2^2 \\
&= \frac{L_g^2}{K}\mathbb{E}\left\|\sum_{j=0}^{s-1}\frac{\gamma_{k(j)}}{B}\sum_{i\in I_t}\nabla_{k(j)}f_i(w_t)\right\|_2^2 \\
&= \frac{L_g^2}{KB^2}\sum_{j=0}^{s-1}\gamma_{k(j)}^2\mathbb{E}\left\|\sum_{i\in I_t}\left(\nabla_{k(j)}f_i(w_t)-\nabla_{k(j)}f(w_t)\right)+B\nabla_{k(j)}f(w_t)\right\|_2^2 \\
&\leq \frac{L_g^2}{KB^2}\sum_{j=0}^{s-1}\gamma_{k(j)}^2\sum_{i\in I_t}\mathbb{E}\left\|\nabla_{k(j)}f_i(w_t)-\nabla_{k(j)}f(w_t)\right\|_2^2+\frac{L_g^2}{K}\sum_{j=0}^{s-1}\gamma_{k(j)}^2\mathbb{E}\left\|\nabla_{k(j)}f(w_t)\right\|_2^2 \\
&\leq \frac{L_g^2}{KB}\sum_{j=0}^{s-1}\gamma_{k(j)}^2\left(M_{k(j)}\mathbb{E}\left\|\nabla_{k(j)}f(w_t)\right\|_2^2+M\right)+\frac{L_g^2}{K}\sum_{j=0}^{s-1}\gamma_{k(j)}^2\mathbb{E}\left\|\nabla_{k(j)}f(w_t)\right\|_2^2 \\
&\leq \frac{L_g^2}{KB}\sum_{k=1}^{K}\gamma_k^2\left(M_k\mathbb{E}\left\|\nabla_k f(w_t)\right\|_2^2+M\right)+\frac{L_g^2}{K}\sum_{k=1}^{K}\gamma_k^2\mathbb{E}\left\|\nabla_k f(w_t)\right\|_2^2, \qquad (4.15)
\end{aligned}
$$

where the first inequality follows from Assumption 4.3.1, the second inequality follows from $\mathbb{E}\left\|\sum_{i=1}^{n}\xi_i\right\|_2^2 \leq \sum_{i=1}^{n}\mathbb{E}\left\|\xi_i\right\|_2^2$ if $\mathbb{E}[\xi_i]=0$, the third inequality follows from Assumption 4.3.2 and the last inequality is because $s \leq K-1$. Combing inequalities (4.13), (4.14) and (4.15), we have:

$$
\begin{aligned}
\mathbb{E}\left[f(w_{t:s+1})\right] \leq\ & \mathbb{E}\left[f(w_{t:s})\right]-\left(\frac{\gamma_{k(s)}}{2}-\frac{L_{k(s)}\gamma_{k(s)}^2}{2}-\frac{L_{k(s)}\gamma_{k(s)}^2M_{k(s)}}{2B}\right)\mathbb{E}\left\|\nabla_{k(s)}f(w_t)\right\|_2^2 \\
&+\frac{\gamma_{k(s)}L_g^2}{2KB}\sum_{k=1}^{K}\gamma_k^2M_k\mathbb{E}\|\nabla_k f(w_t)\|_2^2+\frac{\gamma_{k(s)}L_g^2}{2K}\sum_{k=1}^{K}\gamma_k^2\mathbb{E}\|\nabla_k f(w_t)\|_2^2 \\
&+\frac{L_{k(s)}\gamma_{k(s)}^2M}{2B}+\frac{\gamma_{k(s)}L_g^2M}{2KB}\sum_{k=1}^{K}\gamma_k^2. \qquad (4.16)
\end{aligned}
$$

By summing from $s=0$ to $K-1$, because $w_t=w_{t:0}$ and $w_{t+1}=w_{t:K}$, we can obtain the upper bound of $\mathbb{E}\left[f(w_{t+1})\right]$ as follows:

$$
\begin{aligned}
& \mathbb{E}\left[f(w_{t+1})\right] \\
\leq\ & \mathbb{E}\left[f(w_t)\right] - \sum_{k=1}^{K}\left(\frac{\gamma_k}{2} - \frac{L_k\gamma_k^2}{2} - \frac{L_k\gamma_k^2 M_k}{2B}\right)\mathbb{E}\left\|\nabla_k f(w_t)\right\|_2^2 \\
& + \frac{\sum_{k=1}^{K}\gamma_k L_g^2}{2KB}\sum_{k=1}^{K}\gamma_k^2 M_k \mathbb{E}\|\nabla_k f(w_t)\|_2^2 + \frac{\sum_{k=1}^{K}\gamma_k L_g^2}{2K}\sum_{k=1}^{K}\gamma_k^2 \mathbb{E}\|\nabla_k f(w_t)\|_2^2 \\
& + \sum_{k=1}^{K}\frac{L_k\gamma_k^2 M}{2B} + \frac{\sum_{k=1}^{K}\gamma_k L_g^2 M}{2KB}\sum_{k=1}^{K}\gamma_k^2 \\
\leq\ & \mathbb{E}\left[f(w_t)\right] - \sum_{k=1}^{K}\frac{\gamma_k}{2}\Bigg(1 - L_k\gamma_k - \frac{L_k\gamma_k M_k}{B} - \frac{L_g^2\gamma_k M_k\sum_{k=1}^{K}\gamma_k}{KB} \\
& - \frac{L_g^2\gamma_k\sum_{k=1}^{K}\gamma_k}{K}\Bigg)\mathbb{E}\left\|\nabla_k f(w_t)\right\|_2^2 + \sum_{k=1}^{K}\frac{L_k\gamma_k^2 M}{2B} + \frac{\sum_{k=1}^{K}\gamma_k L_g^2 M}{2KB}\sum_{k=1}^{K}\gamma_k^2. \qquad (4.17)
\end{aligned}
$$

□

Theorem 4.3.1 (Convergence of mGD)**.** *Under Assumptions 4.3.1 and 4.3.2, let* $f_{\inf}$ *denote the minimum value of problem* $f(w)$*,* $\kappa_k = \frac{L_g}{L_k} \leq \kappa$*,* $\gamma_k = \frac{\gamma}{L_k}$*, and* $\sum_{k=1}^{K} q_k \mathbb{E}\left\|\nabla_k f(w_t)\right\|_2^2$ *represents the expectation of* $\mathbb{E}\left\|\nabla_k f(w_t)\right\|_2^2$ *with probability* $q_k = \frac{1/L_k}{\sum_{k=1}^{K}(1/L_k)}$ *for any* k*. As long as* $\gamma_k \leq \min\left\{\frac{1}{8L_k}, \frac{B}{8L_k M_k}\right\}$ *and* $\frac{1}{K}\sum_{k=1}^{K}\gamma_k \leq \min\left\{\frac{1}{2L_g}, \frac{1}{2L_g}\sqrt{\frac{B}{M_g}}\right\}$*, it is guaranteed that:*

$$
\frac{1}{T}\sum_{t=0}^{T-1}\sum_{k=1}^{K} q_k \mathbb{E}\left\|\nabla_k f(w_t)\right\|_2^2 \ \leq\ \frac{8(f(w_0) - f_{\inf})}{T\gamma\sum_{k=1}^{K}\frac{1}{L_k}} + \frac{(4+2\kappa)M\gamma}{B}. \qquad (4.18)
$$

Proof of Theorem 4.3.1: Following Lemma 4.3.1 and defining $\kappa_k = \frac{L_g}{L_k} \leq \kappa$, if γ_k satisfies following inequalities:

$$\begin{aligned} L_k\gamma_k &\leq \frac{1}{8}, && (4.19)\\ \frac{L_k\gamma_k M_k}{B} &\leq \frac{1}{8}, && (4.20)\\ \frac{L_g^2\gamma_k M_k \sum_{k=1}^{K}\gamma_k}{KB} &\leq \frac{1}{4}, && (4.21)\\ \frac{L_g^2\gamma_k \sum_{k=1}^{K}\gamma_k}{K} &\leq \frac{1}{4}, && (4.22) \end{aligned}$$

which are equivalent to $\gamma_k \leq \min\left\{\frac{1}{8L_k}, \frac{B}{8L_kM_k}\right\}$ and $\frac{1}{K}\sum_{k=1}^{K}\gamma_k \leq \min\left\{\frac{1}{2L_g}, \frac{1}{2L_g}\sqrt{\frac{B}{M_g}}\right\}$. Therefore, it holds that:

$$\mathbb{E}\left[f(w_{t+1})\right] \leq \mathbb{E}\left[f(w_t)\right] - \sum_{k=1}^{K}\frac{\gamma_k}{8}\mathbb{E}\left\|\nabla_k f(w_t)\right\|_2^2 + \sum_{k=1}^{K}\frac{(2+\kappa)ML_k\gamma_k^2}{4B}. \quad (4.23)$$

Rearranging the above inequality and summing it from $t=0$ to $T-1$, we have:

$$\frac{1}{8}\sum_{t=0}^{T-1}\sum_{k=1}^{K}\gamma_k\mathbb{E}\left\|\nabla_k f(w_t)\right\|_2^2 \leq f(w_0) - \mathbb{E}\left[f(w_T)\right] + \frac{(2+\kappa)MT}{4B}\sum_{k=1}^{K}L_k\gamma_k^2. \quad (4.24)$$

Because $f(w_T) \geq f_{\inf}$, let $\gamma_k = \frac{\gamma}{L_k}$ and dividing both sides by $\frac{T}{8}\sum_{k=1}^{K}\gamma_k$, we have:

$$\frac{1}{T}\sum_{t=0}^{T-1}\sum_{k=1}^{K}q_k\mathbb{E}\left\|\nabla_k f(w_t)\right\|_2^2 \leq \frac{8(f(w_0)-f_{\inf})}{T\gamma\sum_{k=1}^{K}\frac{1}{L_k}} + \frac{(4+2\kappa)M\gamma}{B}, \quad (4.25)$$

where $q_k = \frac{\frac{1}{L_k}}{\sum_{k=1}^{K}\frac{1}{L_k}}$. We complete the proof. □

Different from Theorem 4.2.1, we use $\sum_{k=1}^{K}q_k\mathbb{E}\left\|\nabla_k f(w_t)\right\|_2^2$ to measure convergence in the chapter. Specially, if $L_k = L_g$ for all k, it is easy to know that $q_k = \frac{1}{K}$ for all k and $\sum_{k=1}^{K}q_k\mathbb{E}\left\|\nabla_k f(w_t)\right\|_2^2 = \frac{1}{K}\mathbb{E}\left\|\nabla f(w_t)\right\|_2^2$. From Theorem 4.3.1, we prove that mGD admits sub-linear convergence rate $O\left(\frac{1}{\sqrt{T}}\right)$ for non-convex problems.

Corollary 4.3.1 (Sub-Linear Convergence Rate of mGD)**.** *Theorem 6.4.2 is satisfied and follow its notations. Suppose* $\frac{1}{8L_k}$ *dominates the upper bound of* γ_k*, and let learning rate* $\gamma = \min\left\{\frac{1}{8}, \sqrt{\frac{B(f(w_0)-f_{\inf})}{TM\sum_{k=1}^{K}\frac{1}{L_k}}}\right\}$*, mGD is guaranteed to converge that:*

$$\frac{1}{T}\sum_{t=0}^{T-1}\sum_{k=1}^{K} q_k \mathbb{E}\left\|\nabla_k f(w_t)\right\|_2^2 \leq \frac{64(f(w_0)-f_{\inf})}{T\sum_{k=1}^{K}\frac{1}{L_k}} + (12+2\kappa)\sqrt{\frac{M(f(w_0)-f_{\inf})}{TB\sum_{k=1}^{K}\frac{1}{L_k}}}. \quad (4.26)$$

Proof of Corollary 4.3.1: Suppose $\frac{1}{8L_k}$ dominates the upper bound of γ_k and:

$$\gamma = \min\left\{\frac{1}{8}, \sqrt{\frac{B(f(w_0)-f_{\inf})}{TM\sum_{k=1}^{K}\frac{1}{L_k}}}\right\}. \quad (4.27)$$

Because $\min_{t\in\{0,\ldots,T-1\}}\sum_{k=1}^{K} q_k \mathbb{E}\left\|\nabla_k f(w_t)\right\|_2^2 \leq \frac{1}{T}\sum_{t=0}^{T-1}\sum_{k=1}^{K} q_k \mathbb{E}\left\|\nabla_k f(w_t)\right\|_2^2$, we have:

$$\begin{aligned}
\frac{1}{T}\sum_{t=0}^{T-1}\sum_{k=1}^{K} q_k \mathbb{E}\left\|\nabla_k f(w_t)\right\|_2^2 &\leq \frac{8(f(w_0)-f_{\inf})}{T\sum_{k=1}^{K}\frac{1}{L_k}}\max\left\{8, \sqrt{\frac{TM\sum_{k=1}^{K}\frac{1}{L_k}}{B(f(w_0)-f_{\inf})}}\right\} \\
&\quad + \frac{(4+2\kappa)M}{B}\sqrt{\frac{B(f(w_0)-f_{\inf})}{TM\sum_{k=1}^{K}\frac{1}{L_k}}} \\
&\leq \frac{64(f(w_0)-f_{\inf})}{T\sum_{k=1}^{K}\frac{1}{L_k}} + (12+2\kappa)\sqrt{\frac{M(f(w_0)-f_{\inf})}{TB\sum_{k=1}^{K}\frac{1}{L_k}}}. \quad (4.28)
\end{aligned}$$

□

So far, we have proved the convergence of mGD method for non-convex problems. When $\beta \neq 0$, we can also prove the convergence of mNAG.

Following [103], we define $p_t = \frac{\beta}{1-\beta}(w_t - w_{t-1} + g_{t-1})$, where $w_{-1} = w_0$, $g_{-1} = 0$, and $g_t = \sum_{k=1}^{K}\gamma_k U_k\left(\frac{1}{B}\sum_{i\in I_t}\nabla_k f_i(w_t)\right)$. Let $z_t = w_t + p_t$, we prove that $\mathbb{E}[f(z_{t+1})]$ is upper bounded at each step in the following Lemma.

Lemma 4.3.2. *Under Assumptions 4.3.1 and 4.3.2, after applying Eq. (4.9) for K micro-steps from $s = 0$ to $s = K - 1$, we have the upper bound of loss $\mathbb{E}[f(z_{t+1})]$ as follows:*

$$\begin{aligned}\mathbb{E}[f(z_{t+1})] \;\leq\; & \mathbb{E}[f(z_t)] - \sum_{k=1}^{K} \frac{\gamma_k}{2(1-\beta)} \Bigg(1 - \frac{L_k\gamma_k}{1-\beta} - \frac{L_k\gamma_k M_k}{(1-\beta)B} - \frac{2L_g^2\gamma_k M_k}{(1-\beta)^2 KB} \sum_{k=1}^{K} \gamma_k \\ & - \frac{2L_g^2\gamma_k}{(1-\beta)^2 K} \sum_{k=1}^{K} \gamma_k \Bigg) \mathbb{E}\left\|\nabla_k f(w_t)\right\|_2^2 + \frac{ML_g^2}{(1-\beta)^3 KB} \sum_{k=1}^{K} \gamma_k \sum_{k=1}^{K} \gamma_k^2 \\ & + \sum_{k=1}^{K} \frac{L_k\gamma_k^2 M}{2(1-\beta)^2 B} + \sum_{k=1}^{K} \frac{L_g^2\gamma_k}{(1-\beta)K} \mathbb{E}\left\|p_t\right\|_2^2 . \end{aligned} \tag{4.29}$$

Proof of Lemma 4.3.2: We define $w_{t:0} = w_t$, and at step $t{:}s$, we have layer index $k(s) = s{+}1$. Thus, we can rewrite Eq. (4.9) for any $s \in \{0, 1, ..., K-1\}$ as follows:

$$\begin{aligned} w_{t:s+1} \;=\; & w_{t:s} - \frac{\gamma_{k(s)}}{B} \sum_{i \in I_t} U_{k(s)} \nabla_{k(s)} f_i(w_t) + \beta \Bigg(w_{t:s} - \frac{\gamma_{k(s)}}{B} \sum_{i \in I_t} U_{k(s)} \nabla_{k(s)} f_i(w_t) \\ & - w_{t:s-1} + \frac{\gamma_{k(s-1)}}{B} \sum_{i \in I_t} U_{k(s-1)} \nabla_{k(s-1)} f_i(w_t) \Bigg), \end{aligned} \tag{4.30}$$

where we let $w_{t:0} = w_{t:-1}$ and $\nabla_{k(-1)} f_{i(-1)}(w_t) = 0$. We also define $p_{t:s}$ as follows:

$$p_{t:s} \;=\; \frac{\beta}{1-\beta} \left(w_{t:s} - w_{t:s-1} + \frac{\gamma_{k(s-1)}}{B} \sum_{i \in I_t} U_{k(s-1)} \nabla_{k(s-1)} f_i(w_t) \right). \tag{4.31}$$

Combining (4.30) and (4.31), we have:

$$w_{t:s+1} + p_{t:s+1} \;=\; w_{t:s} + p_{t:s} - \frac{\gamma_{k(s)}}{(1-\beta)B} \sum_{i \in I_t} U_{k(s)} \nabla_{k(s)} f_i(w_t). \tag{4.32}$$

Let $z_{t:s} = w_{t:s} + p_{t:s}$, according to Assumption 4.3.1, we have the upper bound of $\mathbb{E}[f(z_{t:s+1})]$ as follows:

$$\begin{aligned}
\mathbb{E}[f(z_{t:s+1})] &\leq \mathbb{E}[f(z_{t:s})] - \frac{\gamma_{k(s)}}{(1-\beta)}\mathbb{E}\left\langle \nabla_{k(s)} f(z_{t:s}), \nabla_{k(s)} f(w_t)\right\rangle \\
&\quad + \frac{L_{k(s)}\gamma_{k(s)}^2}{2(1-\beta)^2}\mathbb{E}\left\| \frac{1}{B}\sum_{i\in I_t} \nabla_{k(s)} f_i(w_t)\right\|_2^2 \\
&= \mathbb{E}[f(z_{t:s})] + \frac{L_{k(s)}\gamma_{k(s)}^2}{2(1-\beta)^2}\underbrace{\mathbb{E}\left\| \frac{1}{B}\sum_{i\in I_t} \nabla_{k(s)} f_i(w_t)\right\|_2^2}_{C_3} \\
&\quad - \frac{\gamma_{k(s)}}{2(1-\beta)}\Bigg(\mathbb{E}\left\|\nabla_{k(s)} f(z_{t:s})\right\|_2^2 + \mathbb{E}\left\|\nabla_{k(s)} f(w_t)\right\|_2^2 \\
&\quad - \underbrace{\mathbb{E}\left\|\nabla_{k(s)} f(z_{t:s}) - \nabla_{k(s)} f(w_t)\right\|_2^2}_{C_4}\Bigg).
\end{aligned} \tag{4.33}$$

From (4.14), it is easy to know that the upper bound of C_3 as follows:

$$C_3 \leq \left(1 + \frac{M_{k(s)}}{B}\right)\mathbb{E}\left\|\nabla_{k(s)} f(w_t)\right\|_2^2 + \frac{M}{B}. \tag{4.34}$$

We then obtain the upper bound of C_4:

$$\begin{aligned}
C_4 &\leq \frac{L_g^2}{K}\mathbb{E}\left\|z_{t:s} - w_{t:0}\right\|_2^2 \\
&= \frac{L_g^2}{K}\mathbb{E}\left\|z_{t:s} - z_{t:0} + z_{t:0} - w_{t:0}\right\|_2^2 \\
&\leq \frac{2L_g^2}{(1-\beta)^2 K B^2}\mathbb{E}\left\|\sum_{j=0}^{s-1}\gamma_{k(j)}\sum_{i\in I_t}\nabla_{k(j)} f_i(w_t)\right\|_2^2 + \frac{2L_g^2}{K}\mathbb{E}\left\|p_{t:0}\right\|_2^2 \\
&\leq \frac{2L_g^2}{(1-\beta)^2 K B}\sum_{k=1}^{K}\gamma_k^2\left(M_k\mathbb{E}\left\|\nabla_k f(w_t)\right\|_2^2 + M\right) \\
&\quad + \frac{2L_g^2}{(1-\beta)^2 K}\sum_{k=1}^{K}\gamma_k^2\mathbb{E}\left\|\nabla_k f(w_t)\right\|_2^2 + \frac{2L_g^2}{K}\mathbb{E}\left\|p_{t:0}\right\|_2^2,
\end{aligned} \tag{4.35}$$

where the first inequality follows from $\|a+b\|_2^2 \leq 2\|a\|_2^2 + 2\|b\|_2^2$ and the second inequality follows from inequality (4.15). After combining (4.33), (4.34) and (4.35), we have:

$$\begin{aligned}
\mathbb{E}[f(z_{t:s+1})] \;\leq\; & \mathbb{E}[f(z_{t:s})] - \left(\frac{\gamma_{k(s)}}{2(1-\beta)} - \frac{L_{k(s)}\gamma_{k(s)}^2}{2(1-\beta)^2} - \frac{L_{k(s)}\gamma_{k(s)}^2 M_{k(s)}}{2(1-\beta)^2 B}\right)\mathbb{E}\left\|\nabla_{k(s)} f(w_t)\right\|_2^2 \\
& + \frac{L_g^2 \gamma_{k(s)}}{(1-\beta)^3 KB}\sum_{k=1}^{K}\gamma_k^2 M_k \mathbb{E}\left\|\nabla_k f(w_t)\right\|_2^2 + \frac{L_g^2\gamma_{k(s)}}{(1-\beta)^3 K}\sum_{k=1}^{K}\gamma_k^2 \mathbb{E}\left\|\nabla_k f(w_t)\right\|_2^2 \\
& + \frac{ML_g^2\gamma_{k(s)}}{(1-\beta)^3 KB}\sum_{k=1}^{K}\gamma_k^2 + \frac{L_{k(s)}\gamma_{k(s)}^2 M}{2(1-\beta)^2 B} + \frac{L_g^2\gamma_{k(s)}}{(1-\beta)K}\mathbb{E}\left\|p_t\right\|_2^2 . \qquad (4.36)
\end{aligned}$$

Summing (4.36) from $s=0$ to $K-1$, because $z_{t:0} = z_t$ and $z_{t:K} = z_{t+1}$, we have:

$$\begin{aligned}
\mathbb{E}[f(z_{t+1})] \;\leq\; & \mathbb{E}[f(z_t)] - \sum_{k=1}^{K}\frac{\gamma_k}{2(1-\beta)}\Bigg(1 - \frac{L_k\gamma_k}{1-\beta} - \frac{L_k\gamma_k M_k}{(1-\beta)B} \\
& - \frac{2L_g^2\gamma_k M_k}{(1-\beta)^2 KB}\sum_{k=1}^{K}\gamma_k - \frac{2L_g^2\gamma_k}{(1-\beta)^2 K}\sum_{k=1}^{K}\gamma_k\Bigg)\mathbb{E}\left\|\nabla_k f(w_t)\right\|_2^2 \\
& + \frac{ML_g^2}{(1-\beta)^3 KB}\sum_{k=1}^{K}\gamma_k\sum_{k=1}^{K}\gamma_k^2 + \sum_{k=1}^{K}\frac{L_k\gamma_k^2 M}{2(1-\beta)^2 B} \\
& + \sum_{k=1}^{K}\frac{L_g^2\gamma_k}{(1-\beta)K}\mathbb{E}\left\|p_t\right\|_2^2 . \qquad (4.37)
\end{aligned}$$

□

Lemma 4.3.3. *Under Assumptions 4.3.1 and 4.3.2, after applying Eq. (4.2) from $t=0$ to $T-1$, the following inequality is satisfied that:*

$$\sum_{t=0}^{T-1}\mathbb{E}\left\|p_t\right\|_2^2 \;\leq\; \sum_{k=1}^{K}\frac{\beta^4\gamma_k^2 MT}{(1-\beta)^4 B} + \sum_{k=1}^{K}\left(1+\frac{M_k}{B}\right)\frac{\beta^4\gamma_k^2}{(1-\beta)^4}\sum_{t=0}^{T-1}\mathbb{E}\left\|\nabla_k f(w_t)\right\|_2^2 . \quad (4.38)$$

Proof of Lemma 4.3.3: Let $w_{-1} = w_0$, $g_t = \sum_{k=1}^{K} \frac{\gamma_k}{B} \sum_{i \in I_t} \nabla_k f_i(w_t)$ and $g_{-1} = 0$, we define p_t as follows:

$$p_t = \frac{\beta}{1-\beta} \left(w_t - w_{t-1} + g_{t-1} \right). \tag{4.39}$$

According to the update of mini-batch NAG in Eq. (4.2), it holds that:

$$w_{t+1} = w_t - g_t + \beta \left(w_t - g_t - w_{t-1} + g_{t-1} \right). \tag{4.40}$$

According to the definition of p_t, we have:

$$p_{t+1} = \beta p_t - \frac{\beta^2}{1-\beta} g_t. \tag{4.41}$$

According to Eq. (4.41) and $p_0 = 0$, we know that:

$$\begin{aligned} p_t &= \beta p_{t-1} - \frac{\beta^2}{1-\beta} g_{t-1} \\ &= -\frac{\beta^2}{1-\beta} \sum_{j=0}^{t-1} \beta^{t-1-j} g_j \\ &= -\frac{\beta^2}{1-\beta} \sum_{j=0}^{t-1} \beta^j g_{t-1-j}. \end{aligned} \tag{4.42}$$

Let $\Gamma_{t-1} = \sum_{j=0}^{t-1} \beta^j$, we have:

$$\begin{aligned} \mathbb{E}\|p_t\|_2^2 &= \frac{\beta^4}{(1-\beta)^2} \mathbb{E} \left\| \sum_{j=0}^{t-1} \beta^j g_{t-1-j} \right\|_2^2 \\ &= \frac{\beta^4 \Gamma_{t-1}^2}{(1-\beta)^2} \mathbb{E} \left\| \sum_{j=0}^{t-1} \frac{\beta^j}{\Gamma_{t-1}} g_{t-1-j} \right\|_2^2 \\ &\leq \frac{\beta^4 \Gamma_{t-1}^2}{(1-\beta)^2} \sum_{j=0}^{t-1} \frac{\beta^j}{\Gamma_{t-1}} \mathbb{E} \left\| g_{t-1-j} \right\|_2^2 \\ &= \frac{\beta^4 \Gamma_{t-1}}{(1-\beta)^2} \sum_{j=0}^{t-1} \beta^j \mathbb{E} \left\| g_{t-1-j} \right\|_2^2, \end{aligned} \tag{4.43}$$

where the inequality is from the convexity of $\|\|_2^2$. We can get the upper bound of $\mathbb{E}\|g_t\|_2^2$ as follows:

$$\begin{aligned}
\mathbb{E}\|g_t\|_2^2 &= \mathbb{E}\left\|\sum_{k=1}^{K}\frac{\gamma_k}{B}\sum_{i\in I_t}\nabla_k f_i(w_t)\right\|_2^2 \\
&= \sum_{k=1}^{K}\gamma_k^2\mathbb{E}\left\|\frac{1}{B}\sum_{i\in I_t}\nabla_k f_i(w_t)-\nabla_k f(w_t)+\nabla_k f(w_t)\right\|_2^2 \\
&= \sum_{k=1}^{K}\gamma_k^2\mathbb{E}\left\|\frac{1}{B}\sum_{i\in I_t}\nabla_k f_i(w_t)-\nabla_k f(w_t)\right\|_2^2+\sum_{k=1}^{K}\gamma_k^2\mathbb{E}\|\nabla_k f(w_t)\|_2^2 \\
&\leq \sum_{k=1}^{K}\frac{M\gamma_k^2}{B}+\sum_{k=1}^{K}\left(\gamma_k^2+\frac{M_k\gamma_k^2}{B}\right)\mathbb{E}\|\nabla_k f(w_t)\|_2^2, \qquad (4.44)
\end{aligned}$$

where the third equality follows from $\mathbb{E}\left\langle \frac{1}{B}\sum_{i\in|I_t|}\nabla_k f_i(w_t)-\nabla_k f(w_t), \nabla_k f(w_t)\right\rangle = 0$ and the last inequality follows from Assumption 4.3.2. Combining inequalities (4.43) and (4.44), we have the upper bound of $\mathbb{E}\|(p_t)_k\|_2^2$ as follows:

$$\begin{aligned}
\mathbb{E}\|p_t\|_2^2 &\leq \sum_{k=1}^{K}\frac{\beta^4\gamma_k^2\Gamma_{t-1}}{(1-\beta)^2}\left(\frac{M}{B}\sum_{j=0}^{t-1}\beta^j+\left(1+\frac{M_k}{B}\right)\sum_{j=0}^{t-1}\beta^j\mathbb{E}\|\nabla_k f(w_{t-1-j})\|_2^2\right) \\
&\leq \sum_{k=1}^{K}\frac{\beta^4\gamma_k^2 M}{(1-\beta)^4 B}+\sum_{k=1}^{K}\left(1+\frac{M_k}{B}\right)\frac{\beta^4\gamma_k^2}{(1-\beta)^3}\sum_{j=0}^{t-1}\beta^j\mathbb{E}\|\nabla_k f(w_{t-1-j})\|_2^2, \qquad (4.45)
\end{aligned}$$

where the last inequality follows from $\Gamma_{t-1}=\sum_{j=0}^{t-1}\beta^j=\frac{1-\beta^t}{1-\beta}\leq\frac{1}{1-\beta}$. Summing inequality (4.45) from $t=0$ to $T-1$, we have:

$$\begin{aligned}
\sum_{t=0}^{T-1}\mathbb{E}\|p_t\|_2^2 &\leq \sum_{k=1}^{K}\frac{\beta^4\gamma_k^2 MT}{(1-\beta)^4 B}+\sum_{k=1}^{K}\left(1+\frac{M_k}{B}\right)\frac{\beta^4\gamma_k^2}{(1-\beta)^3}\sum_{t=0}^{T-1}\sum_{j=0}^{t-1}\beta^j\mathbb{E}\|\nabla_k f(w_{t-1-j})\|_2^2 \\
&= \sum_{k=1}^{K}\frac{\beta^4\gamma_k^2 MT}{(1-\beta)^4 B}+\sum_{k=1}^{K}\left(1+\frac{M_k}{B}\right)\frac{\beta^4\gamma_k^2}{(1-\beta)^3}\sum_{t=0}^{T-1}\mathbb{E}\|\nabla_k f(w_t)\|_2^2\sum_{j=t}^{T-1}\beta^{T-1-j} \\
&\leq \sum_{k=1}^{K}\frac{\beta^4\gamma_k^2 MT}{(1-\beta)^4 B}+\sum_{k=1}^{K}\left(1+\frac{M_k}{B}\right)\frac{\beta^4\gamma_k^2}{(1-\beta)^4}\sum_{t=0}^{T-1}\mathbb{E}\|\nabla_k f(w_t)\|_2^2, \qquad (4.46)
\end{aligned}$$

where the last inequality follows from $\sum_{j=t}^{T-1}\beta^{T-1-j}\leq\frac{1}{1-\beta}$ for any $t\in\{0,1,...,T-1\}$. □

Theorem 4.3.2 (Convergence of mNAG)**.** *Under Assumptions 4.3.1 and 4.3.2, let* $f_{\inf}$ *denote the minimum value of problem* $f(w)$*,* $\kappa_k = \frac{L_g}{L_k} \leq \kappa$*,* $\gamma_k = \frac{\gamma}{L_k}$*, and* $\sum_{k=1}^{K} q_k \mathbb{E} \left\| \nabla_k f(w_t) \right\|_2^2$ *represents the expectation of* $\mathbb{E} \left\| \nabla_k f(w_t) \right\|_2^2$ *with probability* $q_k = \frac{1/L_k}{\sum_{k=1}^{K}(1/L_K)}$ *for any* k*. Therefore, as long as:*

$$\gamma_k \leq \min \left\{ \frac{(1-\beta)}{8L_k}, \frac{(1-\beta)B}{8L_k M_k} \right\}, \tag{4.47}$$

$$\frac{1}{K}\sum_{k=1}^{K} \gamma_k \leq \min \left\{ \frac{(1-\beta)^2}{4\beta^2 L_g}, \frac{(1-\beta)^2\sqrt{B}}{4\beta^2 L_g \sqrt{M_g}}, \frac{(1-\beta)\sqrt{B}}{4L_g\sqrt{M_g}}, \frac{(1-\beta)}{4L_g} \right\}, \tag{4.48}$$

it is satisfied that:

$$\frac{1}{T}\sum_{t=0}^{T-1}\sum_{k=1}^{K} q_k \mathbb{E} \left\| \nabla_k f(w_t) \right\|_2^2 \leq \frac{8(1-\beta)(f(w_0) - f_{\inf})}{T\gamma \sum_{k=1}^{K} \frac{1}{L_k}} + \frac{M\gamma}{(1-\beta)B}\left(4 + 2\kappa + \frac{2\kappa}{(1-\beta)}\right). \tag{4.49}$$

Proof of Theorem 4.3.2: Following Lemma 4.3.2 and summing inequality (4.37) from $t = 0$ to $T-1$, we have:

$$\begin{aligned} f_{\inf} \quad \leq \quad & f(w_0) - \sum_{k=1}^{K} \frac{\gamma_k}{2(1-\beta)}\Bigg(1 - \frac{L_k\gamma_k}{1-\beta} - \frac{L_k\gamma_k M_k}{(1-\beta)B} - \frac{2L_g^2\gamma_k M_k}{(1-\beta)^2 KB}\sum_{k=1}^{K}\gamma_k \\ & - \frac{2L_g^2\gamma_k}{(1-\beta)^2 K}\sum_{k=1}^{K}\gamma_k\Bigg)\sum_{t=0}^{T-1}\mathbb{E}\left\|\nabla_k f(w_t)\right\|_2^2 + \frac{ML_g^2 T}{(1-\beta)^3 KB}\sum_{k=1}^{K}\gamma_k\sum_{k=1}^{K}\gamma_k^2 \\ & + \sum_{k=1}^{K}\frac{L_k\gamma_k^2 MT}{2(1-\beta)^2 B} + \sum_{k=1}^{K}\frac{L_g^2\gamma_k}{(1-\beta)K}\sum_{t=0}^{T-1}\mathbb{E}\left\|p_t\right\|_2^2. \end{aligned} \tag{4.50}$$

where we have $z_t = w_0$ and $f(z_T) \geq f_{\inf}$. According to Lemma 4.3.3 and inputting (4.46) in inequality (4.50), the following inequality is satisfied that:

$$\begin{aligned} f_{\inf} \quad \leq \quad & f(w_0) - \sum_{k=1}^{K} \frac{\gamma_k}{2(1-\beta)}\Bigg(1 - \frac{L_k\gamma_k}{1-\beta} - \frac{L_k\gamma_k M_k}{(1-\beta)B} - \frac{2L_g^2\beta^4\gamma_k\sum_{k=1}^{K}\gamma_k}{(1-\beta)^4 K} \\ & - \frac{2L_g^2\beta^4\gamma_k\sum_{k=1}^{K}\gamma_k M_k}{(1-\beta)^4 KB} - \frac{2L_g^2\gamma_k M_k}{(1-\beta)^2 KB}\sum_{k=1}^{K}\gamma_k - \frac{2L_g^2\gamma_k}{(1-\beta)^2 K}\sum_{k=1}^{K}\gamma_k\Bigg)\sum_{t=0}^{T-1}\mathbb{E}\left\|\nabla_k f(w_t)\right\|_2^2 \\ & + \frac{ML_g^2 T}{(1-\beta)^3 KB}\sum_{k=1}^{K}\gamma_k\sum_{k=1}^{K}\gamma_k^2 + \sum_{k=1}^{K}\frac{L_k\gamma_k^2 MT}{2(1-\beta)^2 B} + \sum_{k=1}^{K}\frac{L_g^2\gamma_k\beta^4 MT}{(1-\beta)^5 KB}\sum_{k=1}^{K}\gamma_k^2. \end{aligned} \tag{4.51}$$

Defining $\kappa_k = \frac{L_g}{L_k} \leq \kappa$, if γ_k satisfies following inequalities:

$$\begin{aligned}
\frac{L_k\gamma_k}{1-\beta} &\leq \frac{1}{8}, && (4.52)\\
\frac{L_k\gamma_k M_k}{(1-\beta)B} &\leq \frac{1}{8}, && (4.53)\\
\frac{2L_g^2\beta^4\gamma_k\sum_{k=1}^{K}\gamma_k}{(1-\beta)^4K} &\leq \frac{1}{8}, && (4.54)\\
\frac{2L_g^2\beta^4\gamma_k\sum_{k=1}^{K}M_k\gamma_k}{(1-\beta)^4KB} &\leq \frac{1}{8}, && (4.55)\\
\frac{2L_g^2\gamma_k M_k}{(1-\beta)^2KB}\sum_{k=1}^{K}\gamma_k &\leq \frac{1}{8}, && (4.56)\\
\frac{2L_g^2\gamma_k}{(1-\beta)^2K}\sum_{k=1}^{K}\gamma_k &\leq \frac{1}{8}, && (4.57)
\end{aligned}$$

which are equivalent to:

$$\begin{aligned}
\gamma_k &\leq \min\left\{\frac{(1-\beta)}{8L_k}, \frac{(1-\beta)B}{8L_kM_k}\right\}, && (4.58)\\
\frac{1}{K}\sum_{k=1}^{K}\gamma_k &\leq \min\left\{\frac{(1-\beta)^2}{4\beta^2L_g}, \frac{(1-\beta)^2\sqrt{B}}{4\beta^2L_g\sqrt{M_g}}, \frac{(1-\beta)\sqrt{B}}{4L_g\sqrt{M_g}}, \frac{(1-\beta)}{4L_g}\right\}. && (4.59)
\end{aligned}$$

It holds that:

$$\begin{aligned}
&\sum_{t=0}^{T-1}\sum_{k=1}^{K}\frac{\gamma_k}{8(1-\beta)}\mathbb{E}\left\|\nabla_k f(w_t)\right\|_2^2\\
\leq\ & f(w_0) - f_{\inf} + \frac{MT}{(1-\beta)^2B}\left(\frac{1}{2}\sum_{k=1}^{K}L_k\gamma_k^2 + \frac{L_g^2}{(1-\beta)K}\sum_{k=1}^{K}\gamma_k\sum_{k=1}^{K}\gamma_k^2\right.\\
&\left.+\frac{\beta^4L_g^2}{(1-\beta)^3K}\sum_{k=1}^{K}\gamma_k\sum_{k=1}^{K}\gamma_k^2\right)\\
\leq\ & f(w_0) - f_{\inf} + \frac{MT}{(1-\beta)^2B}\left(\frac{1}{2}\sum_{k=1}^{K}L_k\gamma_k^2 + \frac{1}{4}\sum_{k=1}^{K}L_g\gamma_k^2 + \frac{1}{4(1-\beta)}\sum_{k=1}^{K}L_g\gamma_k^2\right). && (4.60)
\end{aligned}$$

Let $\gamma_k = \frac{\gamma}{L_k}$ and dividing both sides by $\sum_{k=1}^{K} \frac{T}{8(1-\beta)}\gamma_k$, it holds that:

$$\begin{aligned}\frac{1}{T}\sum_{t=0}^{T-1}\sum_{k=1}^{K} q_k \mathbb{E}\left\|\nabla_k f(w_t)\right\|_2^2 &\leq \frac{8(1-\beta)(f(w_0)-f_{\inf})}{T\gamma\sum_{k=1}^{K}\frac{1}{L_k}} \\ &\quad + \frac{M\gamma}{(1-\beta)B}\left(4+2\kappa+\frac{2\kappa}{(1-\beta)}\right). \end{aligned} \tag{4.61}$$

where $q_k = \frac{\frac{1}{L_k}}{\sum_{k=1}^{K}\frac{1}{L_k}}$. □

Similarly, we can easily prove that mNAG is guaranteed to converge for non-convex problems with a sub-linear rate $O\left(\frac{1}{\sqrt{T}}\right)$ as follows:

Corollary 4.3.2 (Sub-Linear Convergence of mNAG)**.** *Theorem 4.3.2 is satisfied and follow its notations, Suppose* $\frac{1-\beta}{8L_k}$ *dominates* γ_k*, if* $\gamma = \min\left\{\frac{1-\beta}{8}, \sqrt{\frac{B(f(w_0)-f_{\inf})}{TM\sum_{k=1}^{K}\frac{1}{L_k}}}\right\}$*, mNAG is guaranteed to converge that:*

$$\begin{aligned}&\min_{t\in\{0,\ldots,T-1\}}\sum_{k=1}^{K} q_k \mathbb{E}\left\|\nabla_k f(w_t)\right\|_2^2 \\ \leq\ & \frac{64(f(w_0)-f_{\inf})}{(1-\beta)T\sum_{k=1}^{K}\frac{1}{L_k}} + \left(8+\frac{1}{(1-\beta)}\left(4+2\kappa+\frac{2\kappa}{(1-\beta)}\right)\right)\sqrt{\frac{M(f(w_0)-f_{\inf})}{TB\sum_{k=1}^{K}\frac{1}{L_k}}}. \end{aligned} \tag{4.62}$$

Proof of Corollary 4.3.2: suppose $\frac{1-\beta}{8L_k}$ dominates the upper bound of γ_k, if:

$$\gamma = \min\left\{\frac{1-\beta}{8}, \sqrt{\frac{B(f(w_0)-f_{\inf})}{TM\sum_{k=1}^{K}\frac{1}{L_k}}}\right\}, \tag{4.63}$$

we can demonstrate that the upper bound of $\min_{t\in\{0,\ldots,T-1\}} \sum_{k=1}^{K} q_k \mathbb{E}\left\|\nabla_k f(w_t)\right\|_2^2$ is guaranteed as follows:

$$
\begin{aligned}
& \min_{t\in\{0,\ldots,T-1\}} \sum_{k=1}^{K} q_k \mathbb{E}\left\|\nabla_k f(w_t)\right\|_2^2 \\
\leq\ & \frac{8(f(w_0)-f_{\inf})}{T\sum_{k=1}^{K}\frac{1}{L_k}} \max\left\{\frac{8}{1-\beta}, \sqrt{\frac{TM\sum_{k=1}^{K}\frac{1}{L_k}}{B(f(w_0)-f_{\inf})}}\right\} \\
& + \frac{M}{(1-\beta)B}\left(4+2\kappa+\frac{2\kappa}{(1-\beta)}\right)\sqrt{\frac{B(f(w_0)-f_{\inf})}{TM\sum_{k=1}^{K}\frac{1}{L_k}}} \\
\leq\ & \frac{64(f(w_0)-f_{\inf})}{(1-\beta)T\sum_{k=1}^{K}\frac{1}{L_k}} + \left(8+\frac{1}{(1-\beta)}\left(4+2\kappa+\frac{2\kappa}{(1-\beta)}\right)\right)\sqrt{\frac{M(f(w_0)-f_{\inf})}{TB\sum_{k=1}^{K}\frac{1}{L_k}}} \qquad (4.64)
\end{aligned}
$$

where the left side follows from $\min_{t\in\{0,\ldots,T-1\}} \sum_{k=1}^{K} q_k \mathbb{E}\left\|\nabla_k f(w_t)\right\|_2^2 \leq \frac{1}{T}\sum_{t=0}^{T-1}\sum_{k=1}^{K} q_k \mathbb{E}\left\|\nabla_k f(w_t)\right\|_2^2$. we complete the proof. □

According to Theorem 4.3.2, we know that the result of Theorem 4.2.1 is a special case of Theorem 4.3.2 when $L_k = L_g$ and $M_k = M_g$.

Corollary 4.3.3 (Convergence when $L_k = L_g$ and $M_k = M_g$)**.** *Suppose Theorem 4.3.2 is satisfied and follow its notations. If $L_k = L_g$, and $M_k = M_g$, $M_C = KM$, we have $\kappa_k = 1$, $\gamma_g = \gamma_k$. As long as the learning rate $\gamma_g \leq \min\left\{\frac{1-\beta}{8L_g}, \frac{B(1-\beta)}{8L_g M_g}, \frac{(1-\beta)\sqrt{B}}{4L_g\sqrt{M_g}}, \frac{(1-\beta)^2\sqrt{B}}{4\beta^2 L_g\sqrt{M_g}}, \frac{(1-\beta)}{4\beta^2 L_g}\right\}$, it is guaranteed that:*

$$
\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\left\|\nabla f(w_t)\right\|_2^2 \leq \frac{8(1-\eta)(f(w_0)-f_{\inf})}{T\gamma_g} + \frac{M_C L_g \gamma_g}{(1-\beta)}\left(6+\frac{2}{1-\beta}\right). \qquad (4.65)
$$

In Corollary 4.3.1 and 4.3.2, we ignore the upper bound of $\frac{1}{K}\sum_{k=1}^{K}\gamma_k$ for simplicity. It can be easily satisfied by making some γ_k small.

4.3.4 Discussions About the Convergence of mNAG

According our fine-grained convergence analysis of gradient-based methods, we take mNAG as an example and gain more insights about the convergence of mNAG for neural networks.

Data Parallelism. Data parallelism is widely used in the training of deep learning models, and linear speedup can be obtained if learning rate and communication can be properly handled. Suppose that $\min_{t\in\{0,\ldots,T_\varepsilon-1\}} \sum_{k=1}^{K} q_k \mathbb{E}\left\|\nabla_k f(w_t)\right\|_2^2 \leq \varepsilon$ is satisfied after optimizing problem $f(w)$ using batch size B after T_ε iterations. Linear speedup means that when batch size scales up by $c \geq 1$ times ($B \to cB$), we can obtain the same convergence guarantee ε after only $\frac{T_\varepsilon}{c}$ iterations ($T_\varepsilon \to \frac{T_\varepsilon}{c}$). From Corollary 4.3.2, if γ is dominated by $\sqrt{\frac{B^2(f(w_0)-f_{\text{inf}})}{TBM\sum_{k=1}^{K}\frac{1}{L_k}}}$, the left term in Eq. (4.62) converges with a rate of $O(\sqrt{\frac{1}{TB}})$. It is guaranteed to converge to the same error as long as TB is fixed. Therefore, we know that when B is scaled up by c times to cB, the problem can converge to the same error after $\frac{T}{c}$ iterations, as long as γ is also scaled up by B times.

Lipschitz Constant Scaled Learning Rate. From Theorem 4.3.2, the learning rate at layer k is computed through $\gamma_k = \frac{\gamma}{L_k}$. It offers us a method to tune K learning rates γ_k for a K-layer neural network simultaneously using just one parameter γ.

Layer-Wise Model Scaling Factor κ_k. Define $\kappa_k = \frac{L_g}{L_k} \geq 1$ as the scaling factor at layer k. Because of the upper bound of $\gamma_k \leq \min\left\{\frac{(1-\beta)}{8L_k}, \frac{(1-\beta)B}{8L_kM_k}\right\}$ in Theorem 4.3.1, we know that designing a layer with larger κ_k can increase the upper bound of learning rate at layer k. In [88], authors show that batch normalization can help to increase κ_k.

Layer-Wise Gradient Variance Factor M_k. Define M_k as the gradient variance factor at layer k, which is dependent on the data and the model, and varies in the process of training. Because of the upper bound of $\gamma_k \leq \min\left\{\frac{(1-\beta)}{8L_k}, \frac{(1-\beta)B}{8L_kM_k}\right\}$ in Theorem 4.3.1, it shows that batch size B can be scaled up as long as $B \leq M_k$. Therefore, a larger M_k helps the algorithm obtain faster speedup. In the following section, we will show that warmup is closely related to M_k.

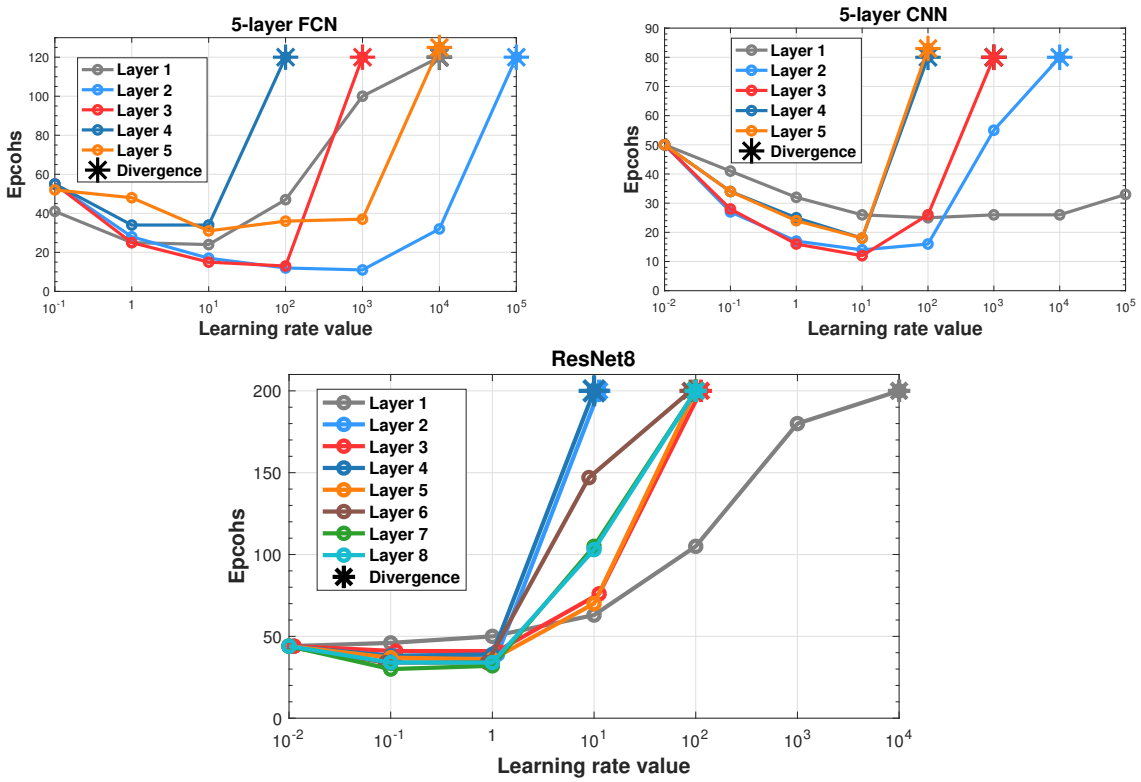


Figure 9: Learning rate upper bound for 5-layer FCN, 5-layer CNN, and 8-layer ResNet.

4.4 Experimental Results

In this section, we conduct experiments to validate our convergence results empirically and demonstrate the superior performance of CLARS method over LARS method. Firstly, we evaluate the necessity of using LARS on training neural networks. Secondly, we verify linear learning rate scaling theoretically and empirically. Thirdly, we propose one hypothesis about the reason of warmup and visualize it. Finally, extensive experiments are conducted to show that CLARS can replace warmup trick completely and converges faster than LARS with fine-tuned warmup steps. All experiments are implemented in PyTorch 1.0 [77] with Cuda v10.0 and performed on a machine with Intel(R) Xeon(R) CPU E5-2683 v4 @ 2.10GHz and 4 Tesla P40 GPUs. We test the performance of CLARS algorithm on fully connected networks, convolutional networks and ResNet.

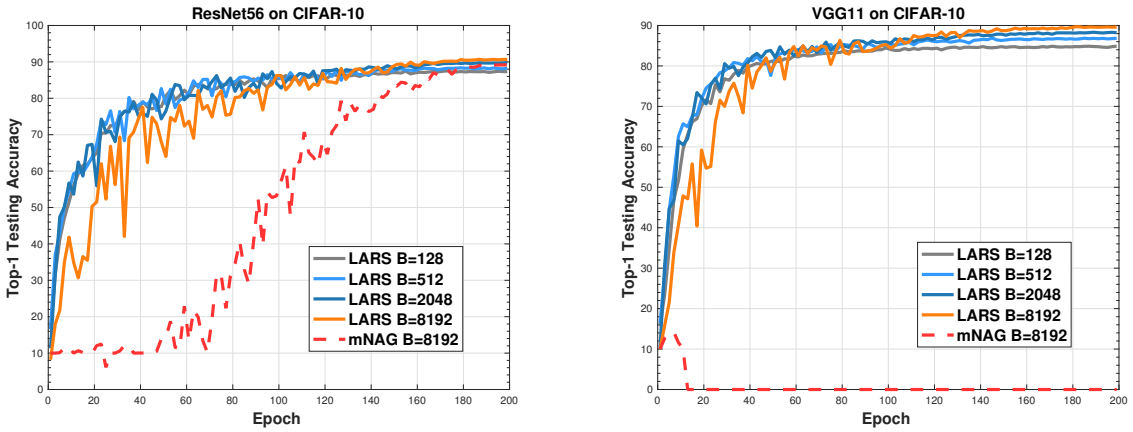


Figure 10: Training loss and Top-1 testing accuracy of training ResNet56 and VGG11 (with batch normalization layer) on CIFAR-10.

4.4.1 Why LARS?

We test the upper bound of learning rate γ_k at each layer on three models: 5-layer FCN, 5-layer CNN (layer details in the Appendix) and ResNet8 (no batch normalization layer) [30]. In the experiments, learning rates are fixed $\gamma_k = 0.01$ for all layers except one which is selected from $\{10^{-2}, 10^{-1}, 1, 10, 10^2, 10^3, 10^4, 10^5\}$. We optimize models using mNAG with $B = 128$ and compare epochs required to achieve the same training loss. Results in Figure We train 5-layer FCN and 5-layer CNN with sigmoid activation on MNIST and count the epochs required reaching training loss 0.03 and 0.02 respectively. We train ResNet8 (no batch normalization layer) on CIFAR-10 and count the epochs required reaching training loss 1.0. * denotes that loss diverges using the corresponding learning rate. 9 demonstrate that the upper bounds of learning rates can vary greatly at different layers. Therefore, it is necessary that each layer has its own learning rate.

From Theorem 4.3.2, we know that the upper bound of learning rate γ_k at each layer is dependent on $\frac{1}{L_k}$. LARS [106] scales the learning rate of each layer adaptively at step t by multiplying $\frac{\|(w_t)_k\|_2}{\|\frac{1}{B}\sum_{i\in I_t}\nabla_k f_i(w_t)\|_2}$ in Eq. (4.3). From Assumption 4.3.1, we can think of LARS as scaling the learning rate at layer k by multiplying the approximation of $\frac{1}{L_k} \approx \frac{\|(w_t)_k\|_2}{\|\nabla_k f(w_t)\|_2}$,

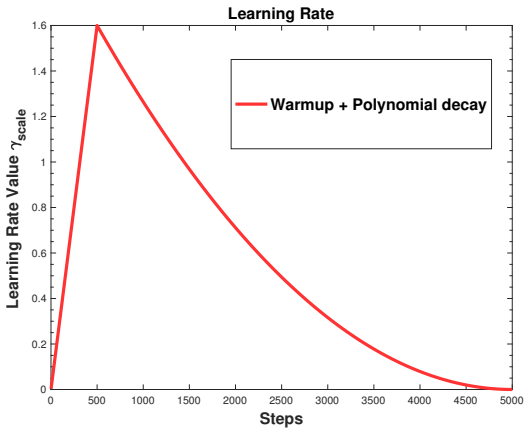


Figure 11: Learning rate schedule.

where we make $v_k = 0$ and $w_t + U_k v_k = 0$. Therefore, the procedure of LARS is consistent with our theoretical analysis in Theorem 4.3.2 that learning rate of layer k is dependent on the Lipschitz constant at this layer $\gamma_k = \frac{\gamma}{L_k}$. We compare LARS with mNAG using a large batch size. Results in Figure 10 demonstrate that LARS converges much faster than mNAG when $B = 8192$. mNAG even diverges in training VGG11 using CIFAR-10. In the experiments, $\gamma_{base} = 0.1$, $B_{base} = 128$, and $\eta = 0.001$ for LARS algorithm.

4.4.2 Linear Learning Rate Scaling

Linear learning rate scaling has been very popular since [28, 52, 58]. However, there is little theoretical understanding of this technique for momentum methods. Based on our analysis in Section 4.3.1, we know that the linear learning rate scaling is from following two reasons:

(I) According to the discussion about Data Parallelism in Section 4.3.4, we know that when B is scaled up by c times to cB, the problem can converge to the same error after $\frac{T}{c}$ iterations, as long as γ is also scaled up by B times.

(II) According to Theorem 4.3.2, as long as $\frac{(1-\beta)B}{8L_k M_k}$ dominates the upper bound of the learning rate γ_k at layer k, its upper bound scales linearly with the batch size B.

The second case requires that $\frac{B}{M_k}$ to be very small. The layer-wise gradient variance factor M_k is closely related to both model and data. In [93], authors find that different models usually have different maximum useful batch size. The variance factor M_k is highly dependent on the dataset and close to the gradient diversity in [104]. We can draw the same conclusion as [104] that mNAG admits better speedup on problems with higher gradient diversity.

In Figure 10, we train ResNet56 [30] and VGG11 with batch normalization layer [41, 95] on CIFAR-10 [53] for 200 epochs. We use LARS optimizer with gradual warmup (20 epochs) and polynomial learning rate decay as [106], which is also visualized in Figure 11. We scale up the batch size from 128 to 8192 and employ the linear learning rate scaling. Results in Figure 10 show that the convergence rates of LARS with batch size from 128 to 8192 are similar and the linear speedup is guaranteed when the computations are parallelized on multiple devices. Because the learning rate schedule is tuned for large-batch training, we may observe accuracy improvements when the batch size scales up.

4.4.3 One Hypothesis About Warmup

The gradual warmup was essential for large-batch deep learning optimization because linearly scaled γ_{scale} can be so large that the loss cannot converge in early epochs [28]. In the gradual warmup, γ_{scale} is replaced with a small value at the beginning and increased back gradually after a few epochs.

According to our analysis in Theorem 4.3.2, we guess that the gradual warmup is to simulate the function of $\frac{1}{M_k}$ in the upper bound of learning rate. We train 5-layer FCN, 5-layer CNN on MNIST [57] and ResNet8 on CIFAR-10 using mNAG for 50 epochs. Constant learning rate 0.001 is used for all layers and batch size $B = 128$. After each epoch, we approximate the gradient variance factor M_k by computing the ratio of $\frac{1}{n}\sum_{i=1}^{n} \|\nabla_k f_i(w_t)\|_2^2$ to $\|\frac{1}{n}\sum_{i=1}^{n} \nabla_k f_i(w_t)\|_2^2$ on training data. Figure 12 presents the variation of M_k at each layers. It is obvious that M_k of top layers are larger than other layers. Thus, smaller learning rates should be used on top layers at early epochs. Our observation matches [27] that freezing fully connected layers at early epochs allows for comparable performance with warmup.

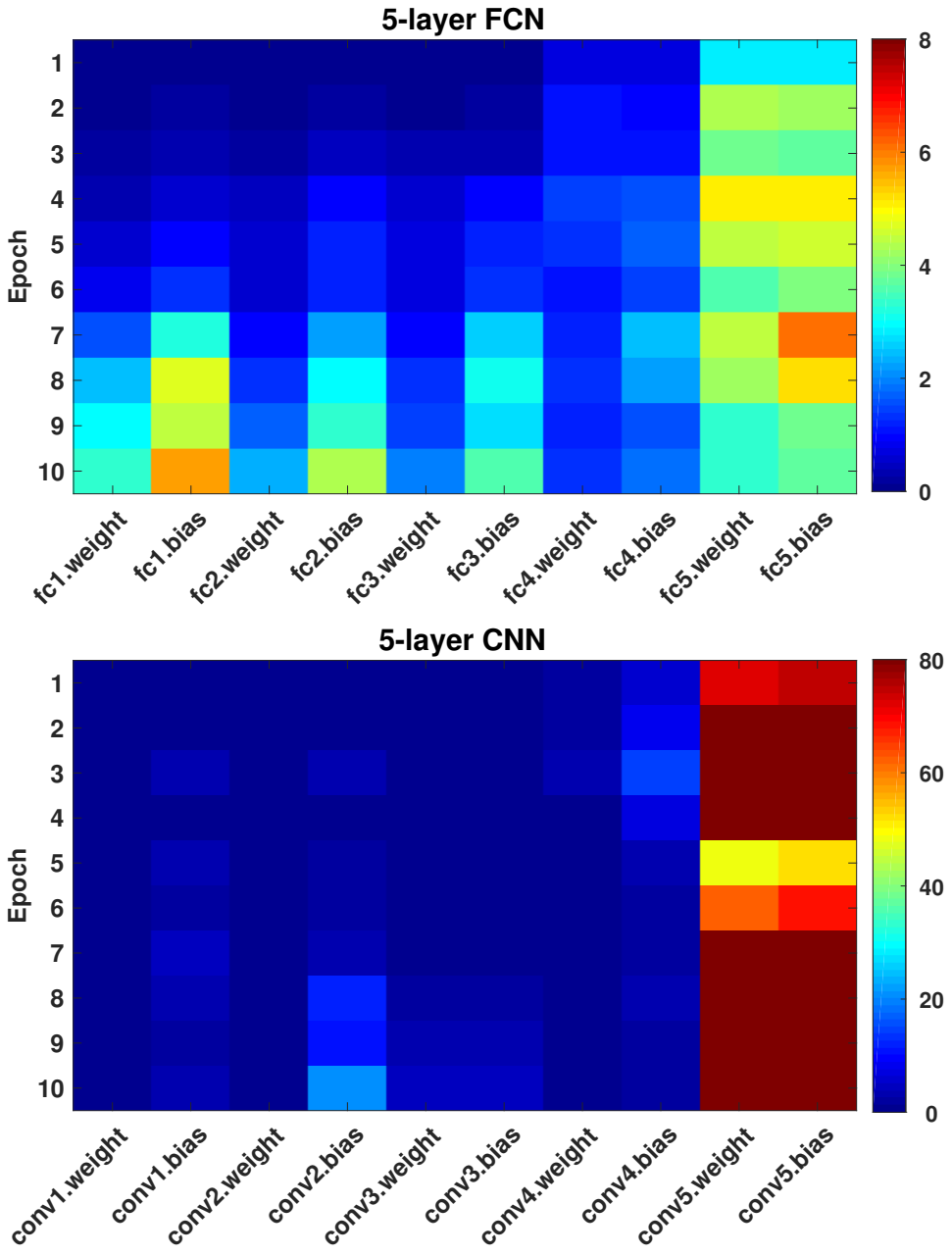


Figure 12: Variation of variance for 10 epochs. We train 5-layer FCN and 5-layer CNN with sigmoid activation on MNIST.

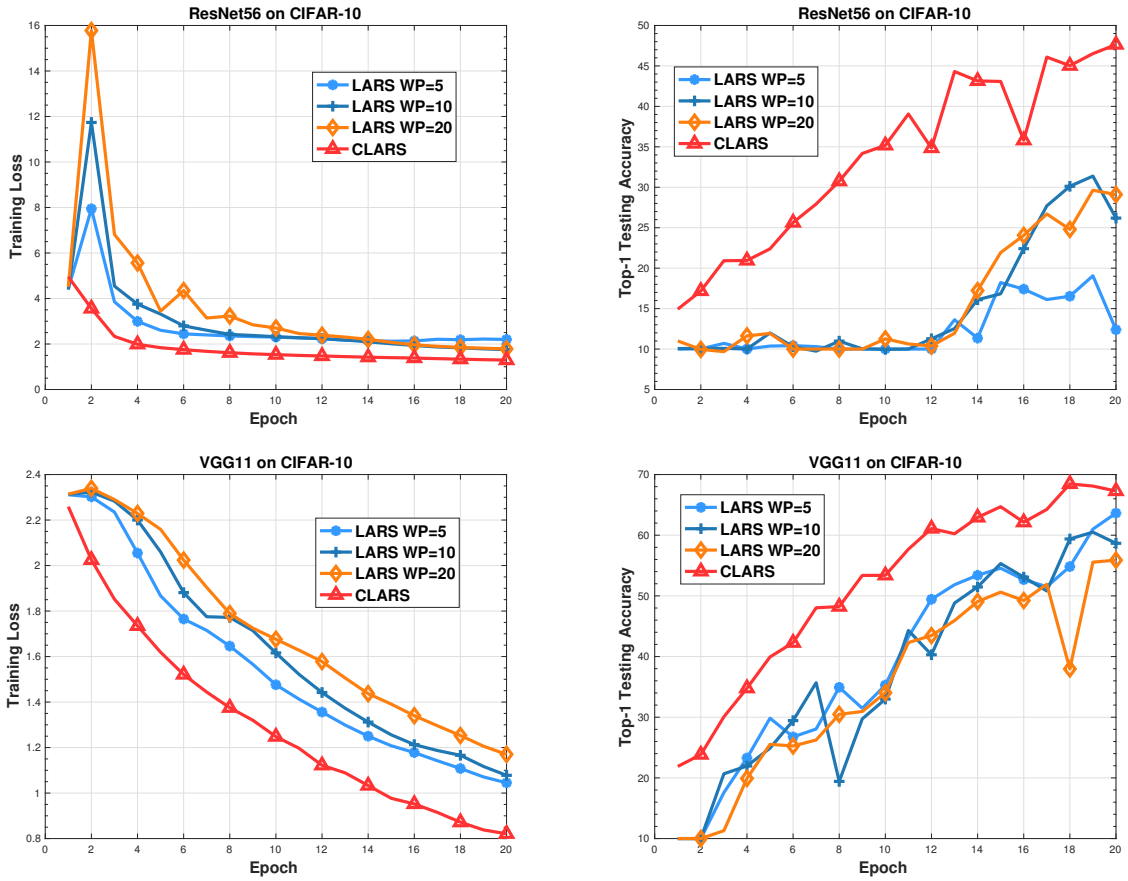


Figure 13: Comparison between LARS (with gradual warmup) and CLARS algorithm.

4.4.4 Warmup is Not Necessary

We evaluate the proposed Algorithm 8 by conducting extensive experiments. To reduce the time consumption in computing M_k, we approximate it using $M_k \approx \frac{\left\|\frac{1}{B}\sum_{i\in I_t}\nabla_k f_i(w_t)\right\|_2^2}{\frac{1}{|J_t|}\sum_{j\in J_t}\|\nabla_k f_j(w_t)\|_2^2}$, where $|J_t| = 512$. The numerator is known after the gradient computation, and the denominator is obtained in a small size. Since $|J_t| \ll B$, the computational time of approximating M_k can be ignored when the computation is amortized on multiple devices. In Figure 13, we make a comparison between LARS (with gradual warmup) and the proposed CLARS algorithm. We train ResNet56 and VGG11 (with batch normalization layer) on CIFAR-10 with batch size $B = 8192$ for 20 epochs. Standard data preprocessing techniques are used as

in [30]. For LARS with the gradual warmup, we test three warmup epochs $\{5, 10, 20\}$ and keep $\gamma_{scale} = 6.4$ after the warmup. For CLARS, we keep $\gamma_{scale} = 6.4$ for 20 epochs. η is tuned from $\{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\}$ for both methods Visualization in Figure 13 shows that CLARS always outperforms LARS by a large margin. Results demonstrate that warmup is not necessary in large-batch deep learning training and CLARS is a better option for practical implementation.

We evaluate CLARS algorithm by training ResNet50, DenseNet121, and MobileNetv2 on ImageNet [18]. Because there are not enough GPUs to compute 16384 gradients at one time, we set batch size $B = 512$ and accumulate the gradients for 32 steps before updating the model as [106]. Following the official implementation[1], we set $\eta = 10^{-3}$ for LARS and adjust the learning rate using 5-epoch warmup and polynomial decay. For CLARS, there is no warmup and we set $\eta = 10^{-2}$ (LARS always diverges with this value). We train ResNet50, DenseNet121 for 90 epochs with batch size $B = 16384$ and $\gamma_{scale} = 25.0$. MobileNetv2 is trained for 150 epochs with batch size $B = 16384$ and $\gamma_{scale} = 6.0$.

Experimental results in Figure 14 present that CLARS algorithm always converges much faster than the state-of-the-art large-batch optimizer LARS on advanced neural networks. Besides, CLARS can obtain better test error than LARS.

[1] `https://github.com/tensorflow/models/blob/master/official/resnet/resnet_run_loop.py`

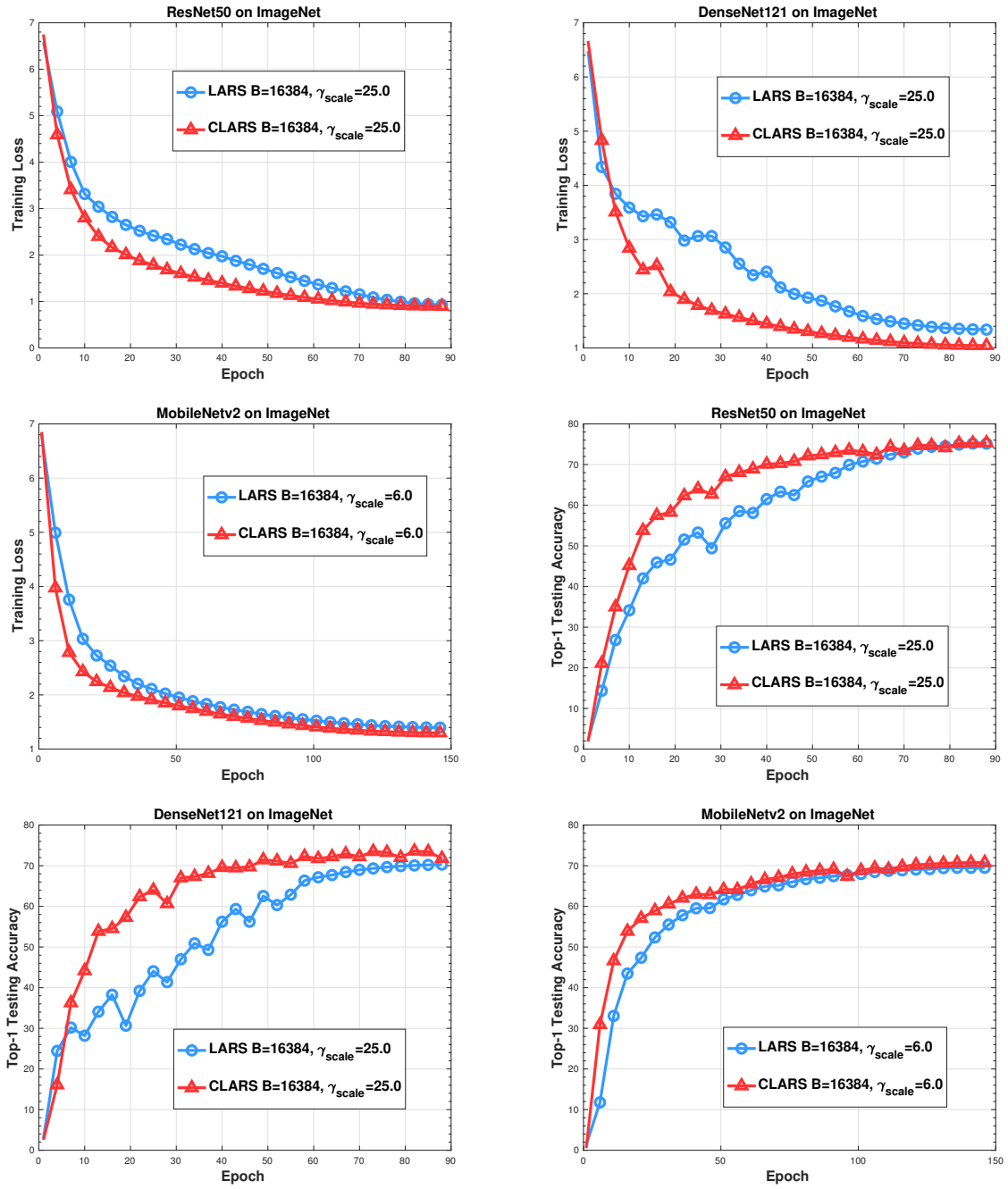


Figure 14: Comparison between LARS and CLARS training ResNet50, DenseNet121, and MobileNetv2 on ImageNet.

5.0 Decoupled Parallel Backpropagation with Convergence Guarantee

5.1 Motivation

We have witnessed a series of breakthroughs in computer vision using deep convolutional neural networks [56]. Most neural networks are trained using stochastic gradient descent (SGD) or its variants in which the gradients of the networks are computed by backpropagation algorithm [85]. As shown in Figure 24, the backpropagation algorithm consists of two processes, the forward pass to compute prediction and the backward pass to compute gradient and update the model. After computing prediction in the forward pass, backpropagation algorithm requires propagating error gradients from the top (output layer) all the way back to the bottom (input layer). Therefore, in the backward pass, all layers, or more generally, modules, of the network are locked until their dependencies have executed.

The backward locking constrains us from updating models in parallel and fully leveraging the computing resources. It has been shown in practice [54, 95, 96, 30, 37] and in theory [21, 99, 8] that depth is one of the most critical factors contributing to the success of deep learning. From AlexNet with 8 layers [54] to ResNet-101 with more than one hundred layers [30], the forward and backward time grow from (4.31ms and 9.58ms) to (53.38ms and 103.06ms) when we train the networks on Titan X with the input size of $16 \times 3 \times 224 \times 224$ [45]. Therefore, parallelizing the backward pass can greatly reduce the training time when the backward time is about twice of the forward time. We can easily split a deep neural network into modules like Figure 15 and distribute them across multiple GPUs. Each module is a stack of layers. Backpropagation algorithm requires running forward pass (from 1 to 3) and backward pass (from 4 to 6) in sequential order. For example, module A cannot perform step 6 before receiving δ_A^t which is an output of step 5 in module B. However, because of the backward locking, all GPUs are idle before receiving error gradients from dependent modules in the backward pass.

There have been several algorithms proposed for breaking the backward locking. For example, [42, 15] proposed to remove the lockings in backpropagation by employing addi-

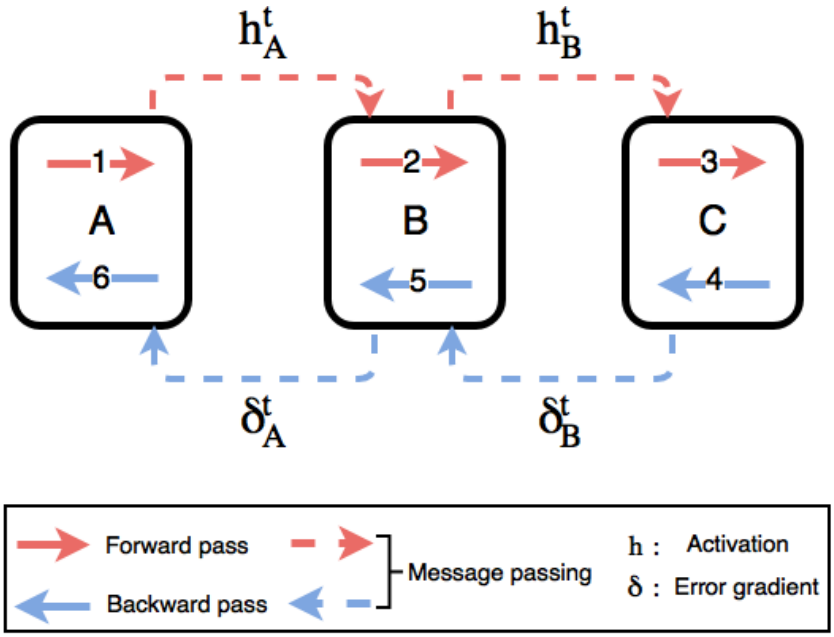


Figure 15: Procedure of the backpropagation algorithm.

tional neural networks to approximate error gradients. In the backward pass, all modules use the synthetic gradients to update weights of the model without incurring any delay. [76, 6] broke the local dependencies between successive layers and made all hidden layers receive error information from the output layer directly. In [11, 98], the authors loosened the exact connections between layers by introducing auxiliary variables. In each layer, they imposed equality constraint between the auxiliary variable and activation, and optimized the new problem using Alternating Direction Method which is easy to parallel. However, for the convolutional neural network, the performances of all above methods are much worse than backpropagation algorithm when the network is deep.

We focus on breaking the backward locking in backpropagtion algorithm for training feedforward neural networks, such that we can update models in parallel without loss of accuracy. The main contributions of our work are as follows:

- Firstly, we decouple the backpropagation using delayed gradients in Section 6.3 such that all modules of the network can be updated in parallel without backward locking.

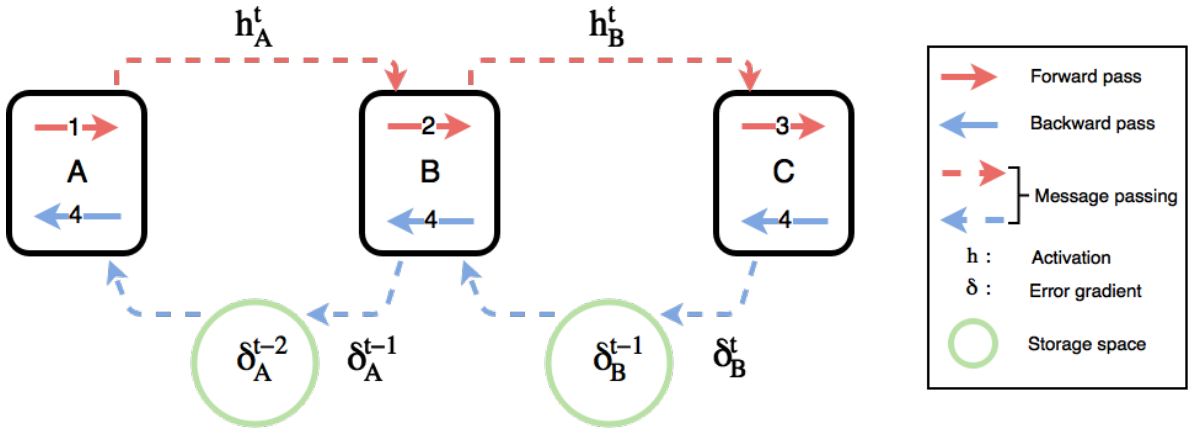


Figure 16: Procedure of the decoupled parallel backpropagation algorithm.

- Then, we propose two stochastic algorithms using decoupled parallel backpropagation in Section 6.3 for deep learning optimization.
- We also provide convergence analysis for the proposed method in Section 6.4 and prove that it guarantees convergence to critical points for the non-convex problem.
- Finally, we perform experiments for training deep convolutional neural networks in Section 6.5, experimental results verifying that the proposed method can significantly speed up the training without loss of accuracy.

5.2 Preliminaries

We begin with a brief overview of the backpropagation algorithm for the optimization of neural networks. Suppose that we want to train a feedforward neural network with L layers, each layer taking an input h_{l-1} and producing an activation $h_l = F_l(h_{l-1}; w_l)$ with weight w_l. Letting d be the dimension of weights in the network, we have $w = [w_1, w_2, ..., w_L] \in \mathbb{R}^d$. Thus, the output of the network can be represented as $h_L = F(h_0; w)$, where h_0 denotes the input data x. Taking a loss function f, model parameter w and data pair (x, y), the training

problem in this section can be represented as follows and we use $f(w)$ for simplicity in the following context:

$$\min_{w=[w_1,...,w_L]} f(F(x;w),y). \tag{5.1}$$

Gradients based methods are widely used for deep learning optimization [84, 78, 33, 50]. In iteration t, we put a data sample $x_{i(t)}$ into the network, where $i(t)$ denotes the index of the sample. According to stochastic gradient descent (SGD), we update the weights of the network through:

$$w_l^{t+1} = w_l^t - \gamma_t \left[\nabla f_{l,x_{i(t)}}(w^t)\right]_l, \ \forall l \in \{1,2,...,L\}. \tag{5.2}$$

where γ_t is the learning rate and $\nabla f_{l,x_{i(t)}}(w^t) \in \mathbb{R}^d$ is the gradient of the loss function (5.1) with respect to the weights at layer l and data sample $x_{i(t)}$, all the coordinates in other than layer l are 0. We always utilize backpropagation algorithm to compute the gradients [85]. The backpropagation algorithm consists of two passes of the network: in the forward pass, the activations of all layers are calculated from $l = 1$ to L as follows:

$$h_l^t = F_l(h_{l-1}^t; w_l); \tag{5.3}$$

in the backward pass, we apply chain rule for gradients and repeatedly propagate error gradients through the network from the output layer $l = L$ to the input layer $l = 1$:

$$\frac{\partial f(w^t)}{\partial w_l^t} = \frac{\partial h_l^t}{\partial w_l^t}\frac{\partial f(w^t)}{\partial h_l^t}, \tag{5.4}$$

$$\frac{\partial f(w^t)}{\partial h_{l-1}^t} = \frac{\partial h_l^t}{\partial h_{l-1}^t}\frac{\partial f(w^t)}{\partial h_l^t}, \tag{5.5}$$

where we let $\nabla f_{l,x_{i(t)}}(w^t) = \frac{\partial f(w^t)}{\partial w_l^t}$. From equations (5.4) and (5.5), it is obvious that the computation in layer l is dependent on the error gradient $\frac{\partial f(w^t)}{\partial h_l^t}$ from layer $l+1$. Therefore, the backward locking constrains all layers from updating before receiving error gradients from the dependent layers. When the network is very deep or distributed across multiple resources, the backward locking is the main bottleneck in the training process.

5.3 Decoupled Parallel Backpropagation

In this section, we propose to decouple the backpropagation algorithm using delayed gradients (DDG). Suppose we split a L-layer feedforward neural network to K modules, such that the weights of the network are divided into K groups. Therefore, we have $w = [w_{\mathcal{G}(1)}, w_{\mathcal{G}(2)}, ..., w_{\mathcal{G}(K)}]$ where $\mathcal{G}(k)$ denotes layer indices in the group k.

5.3.1 Backpropagation Using Delayed Gradients

In iteration t, data sample $x_{i(t)}$ is input to the network. We run the forward pass from module $k = 1$ to $k = K$. In each module, we compute the activations in sequential order as equation (5.3). In the backward pass, all modules except the last one have delayed error gradients in store such that they can execute the backward computation without locking. The last module updates with the up-to-date gradients. In particular, module k keeps the stale error gradient $\frac{\partial f(w^{t-K+k})}{\partial h_{L_k}^{t-K+k}}$, where L_k denotes the last layer in module k. Therefore, the backward computation in module k is as follows:

$$\frac{\partial f(w^{t-K+k})}{\partial w_l^{t-K+k}} = \frac{\partial h_l^{t-K+k}}{\partial w_l^{t-K+k}} \frac{\partial f(w^{t-K+k})}{\partial h_l^{t-K+k}}, \tag{5.6}$$

$$\frac{\partial f(w^{t-K+k})}{\partial h_{l-1}^{t-K+k}} = \frac{\partial h_l^{t-K+k}}{\partial h_{l-1}^{t-K+k}} \frac{\partial f(w^{t-K+k})}{\partial h_l^{t-K+k}}. \tag{5.7}$$

where $\ell \in \mathcal{G}(k)$. Meanwhile, each module also receives error gradient from the dependent module for further computation. From (5.6) and (5.7), we can know that the stale error gradients in all modules are of different time delay. From module $k = 1$ to $k = K$, their corresponding time delays are from $K - 1$ to 0. Delay 0 indicates that the gradients are up-to-date. In this way, we break the backward locking and achieve parallel update in the backward pass. Figure 16 shows an example of the decoupled backpropagation, where error gradients $\delta := \frac{\partial f(w)}{\partial h}$. We split a multi-layer feedforward neural network into three modules (A, B and C), where each module is a stack of layers. After executing the forward pass (from 1 to 3) to predict, our proposed method allows all modules to run backward pass (4) using delayed gradients without locking. Particularly, module A can perform the backward pass

Table 3: Comparisons of computation time when the network is sequentially distributed across K GPUs.

Method	Computation Time
Backpropagation	$\mathcal{T}_F + \mathcal{T}_B$
DDG	$\mathcal{T}_F + \frac{\mathcal{T}_B}{K}$

using the stale error gradient δ_A^{t-2}. Meanwhile, It also receives δ_A^{t-1} from module B for the update of the next iteration.

5.3.2 Speedup of Decoupled Parallel Backpropagation

When $K = 1$, there is no time delay and the proposed method is equivalent to the backpropagation algorithm. When $K \neq 1$, we can distribute the network across multiple GPUs and fully leverage the computing resources. Table 3 lists the computation time when we sequentially allocate the network across K GPUs. $\mathcal{T}_F$ and $\mathcal{T}_B$ denote the forward and backward time for backpropagation algorithm. When $\mathcal{T}_F$ is necessary to compute accurate predictions, we can accelerate the training by reducing the backward time. Because $\mathcal{T}_B$ is much large than $\mathcal{T}_F$, we can achieve huge speedup even K is small.

Relation to model parallelism: Model parallelism usually refers to filter-wise parallelism [101]. For example, we split a convolutional layer with N filters into two GPUs, each part containing $\frac{N}{2}$ filters. Although the filter-wise parallelism accelerates the training when we distribute the workloads across multiple GPUs, it still suffers from the backward locking. We can think of DDG algorithm as layer-wise parallelism. It is also easy to combine filter-wise parallelism with layer-wise parallelism for further speedup.

Algorithm 9 SGD-DDG

Require:

Initial weights $w^0 = [w^0_{\mathcal{G}(1)}, ..., w^0_{\mathcal{G}(K)}] \in \mathbb{R}^d$;

learning rate sequence $\{\gamma_t\}$;

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

2: **for** $k = 1, \ldots, K$ **in parallel do**

3: Compute delayed gradient:

$$g^t_k \leftarrow \left[\nabla f_{\mathcal{G}(k), x_{i(t-K+k)}}\left(w^{t-K+k}\right)\right]_{\mathcal{G}(k)};$$

4: Update weights:

$$w^{t+1}_{\mathcal{G}(k)} \leftarrow w^t_{\mathcal{G}(k)} - \gamma_t \cdot g^t_k;$$

5: **end for**

6: **end for**

5.3.3 Stochastic Methods Using Delayed Gradients

After computing the gradients of the loss function with respect to the weights of the model, we update the model using delayed gradients. Letting $\nabla f_{\mathcal{G}(k), x_{i(t-K+k)}}\left(w^{t-K+k}\right) :=$

$$\begin{cases} \sum\limits_{l \in \mathcal{G}(k)} \frac{\partial f(w^{t-K+k})}{\partial w_l^{t-K+k}} & \text{if } t-K+k \geq 0 \\ 0 & \text{otherwise} \end{cases}, \tag{5.8}$$

for any $k \in \{1, 2, ..., K\}$, we update the weights in module k following SGD:

$$w^{t+1}_{\mathcal{G}(k)} = w^t_{\mathcal{G}(k)} - \gamma_t [\nabla f_{\mathcal{G}(k), x_{i(t-K+k)}}\left(w^{t-K+k}\right)]_{\mathcal{G}(k)}. \tag{5.9}$$

where γ_t denotes learning rate. Different from SGD, we update the weights with delayed gradients. Besides, the delayed iteration $(t-K+k)$ for group k is also deterministic. We summarize the proposed method in Algorithm 11.

Moreover, we can also apply the delayed gradients to other variants of SGD, for example Adam in Algorithm 10. In each iteration, we update the weights and moment vectors with delayed gradients. We analyze the convergence for Algorithm 9 in Section 6.4, which is the basis of analysis for other methods.

Algorithm 10 Adam-DDG

Require:

Initial weights: $w^0 = [w^0_{\mathcal{G}(1)}, ..., w^0_{\mathcal{G}(K)}] \in \mathbb{R}^d$;

learning rate: γ; Constant $\epsilon = 10^{-8}$;

Exponential decay rates: $\beta_1 = 0.9$ and $\beta_2 = 0.999$;

First moment vector: $m^0_{\mathcal{G}(k)} \leftarrow 0, \forall k \in \{1, 2, ..., K\}$;

Second moment vector: $v^0_{\mathcal{G}(k)} \leftarrow 0, \forall k \in \{1, 2, ..., K\}$;

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

  for $k = 1, \ldots, K$ **in parallel do**

    Compute delayed gradient:

$$g^t_k \leftarrow \left[\nabla f_{\mathcal{G}(k), x_{i(t-K+k)}}\left(w^{t-K+k}\right)\right]_{\mathcal{G}(k)};$$

    Update biased first moment estimate:

$$m^{t+1}_{\mathcal{G}(k)} \leftarrow \beta_1 \cdot m^t_{\mathcal{G}(k)} + (1-\beta_1) \cdot g^t_k;$$

    Update biased second moment estimate:

$$v^{t+1}_{\mathcal{G}(k)} \leftarrow \beta_2 \cdot v^t_{\mathcal{G}(k)} + (1-\beta_2) \cdot (g^t_k)^2;$$

    Compute bias-correct first moment estimate:

$$\hat{m}^{t+1}_{\mathcal{G}(k)} \leftarrow m^{t+1}_{\mathcal{G}(k)} / (1-\beta_1^{t+1});$$

    Compute bias-correct second moment estimate:

$$\hat{v}^{t+1}_{\mathcal{G}(k)} \leftarrow v^{t+1}_{\mathcal{G}(k)} / (1-\beta_2^{t+1});$$

    Update weights:

$$w^{t+1}_{\mathcal{G}(k)} \leftarrow w^t_{\mathcal{G}(k)} - \gamma \cdot \hat{m}^{t+1}_{\mathcal{G}(k)} / \left(\sqrt{\hat{v}^{t+1}_{\mathcal{G}(k)}} + \epsilon\right);$$

  end for

end for

5.4 Convergence Analysis

In this section, we establish the convergence guarantees to critical points for Algorithm 9 when the problem is non-convex. Analysis shows that our method admits similar convergence rate to vanilla stochastic gradient descent [10]. Throughout this chapter, we make the following commonly used assumptions:

Assumption 5.4.1. ***(Lipschitz-continuous gradient)*** *The gradient of $f(w)$ is Lipschitz continuous with Lipschitz constant $L > 0$, such that $\forall w, v \in \mathbb{R}^d$:*

$$\|\nabla f(w) - \nabla f(v)\|_2 \leq L\|w - v\|_2. \tag{5.10}$$

Assumption 5.4.2. ***(Bounded variance)*** *To bound the variance of the stochastic gradient, we assume the second moment of the stochastic gradient is upper bounded, such that there exists constant $M \geq 0$, for any sample x_i and $\forall w \in \mathbb{R}^d$:*

$$\|\nabla f_{x_i}(w)\|_2^2 \quad \leq \quad M. \tag{5.11}$$

Because of the unnoised stochastic gradient $\mathbb{E}\left[\nabla f_{x_i}(w)\right] = \nabla f(w)$ and the equation regarding variance $\mathbb{E}\,\|\nabla f_{x_i}(w) - \nabla f(w)\|_2^2 = \mathbb{E}\|\nabla f_{x_i}(w)\|_2^2 - \|\nabla f(w)\|_2^2$, the variance of the stochastic gradient is guaranteed to be less than M.

Under Assumption 5.4.1 and 5.4.2, we obtain the following lemma about the sequence of objective functions.

Lemma 5.4.1. *Assume Assumption 5.4.1 and 5.4.2 hold. We let $\sigma := \max_t \frac{\gamma_{\max\{0,t-K+1\}}}{\gamma_t}$ and $M_K = KM + \sigma K^4 M$. The iterations in Algorithm 9 satisfy the following inequality, for all $t \in \mathbb{N}$:*

$$\mathbb{E}\left[f(w^{t+1})\right] - f(w^t) \leq -\frac{\gamma_t}{2}\left\|\nabla f(w^t)\right\|_2^2 + \gamma_t^2 L M_K. \tag{5.12}$$

Proof of Lemma 5.4.1: Because the gradient of $f(w)$ is Lipschitz continuous in Assumption 5.4.1, the following inequality holds that:

$$f(w^{t+1}) \quad \leq \quad f(w^t) + \nabla f(w^t)^T\left(w^{t+1} - w^t\right) + \frac{L}{2}\left\|w^{t+1} - w^t\right\|_2^2. \tag{5.13}$$

From the update rule in Algorithm 9, we take expectation on both sides and obtain the upper bound for $\mathbb{E}\left[f(w^{t+1})\right]$:

$$
\begin{aligned}
\mathbb{E}\left[f(w^{t+1})\right] &\leq f(w^t) - \gamma_t \mathbb{E}\left[\nabla f(w^t)^T \left(\sum_{k=1}^{K} \nabla f_{\mathcal{G}(k),x_{i(t-K+k)}}\left(w^{t-K+k}\right)\right)\right] \\
&\quad + \frac{L\gamma_t^2}{2}\mathbb{E}\left\|\sum_{k=1}^{K} \nabla f_{\mathcal{G}(k),x_{i(t-K+k)}}(w^{t-K+k})\right\|_2^2 \\
&\leq f(w^t) - \gamma_t \sum_{k=1}^{K} \nabla f(w^t)^T \left(\nabla f_{\mathcal{G}(k)}\left(w^{t-K+k}\right) + \nabla f_{\mathcal{G}(k)}\left(w^t\right) - \nabla f_{\mathcal{G}(k)}\left(w^t\right)\right) \\
&\quad + \frac{L\gamma_t^2}{2}\mathbb{E}\left\|\sum_{k=1}^{K} \nabla f_{\mathcal{G}(k),x_{i(t-K+k)}}(w^{t-K+k}) - \nabla f(w^t) + \nabla f(w^t)\right\|_2^2 \\
&= f(w^t) - \gamma_t \left\|\nabla f(w^t)\right\|_2^2 - \gamma_t \sum_{k=1}^{K} \nabla f(w^t)^T \left(\nabla f_{\mathcal{G}(k)}\left(w^{t-K+k}\right) - \nabla f_{\mathcal{G}(k)}\left(w^t\right)\right) \\
&\quad + \frac{L\gamma_t^2}{2}\left\|\nabla f(w^t)\right\|_2^2 + \frac{L\gamma_t^2}{2}\mathbb{E}\left\|\sum_{k=1}^{K} \nabla f_{\mathcal{G}(k),x_{i(t-K+k)}}(w^{t-K+k}) - \nabla f(w^t)\right\|_2^2 \\
&\quad + L\gamma_t^2 \sum_{k=1}^{K} \nabla f(w^t)^T \left(\nabla f_{\mathcal{G}(k)}\left(w^{t-K+k}\right) - \nabla f_{\mathcal{G}(k)}\left(w^t\right)\right) \\
&= f(w^t) - \left(\gamma_t - \frac{L\gamma_t^2}{2}\right)\left\|\nabla f(w^t)\right\|_2^2 \\
&\quad \underbrace{+ \frac{L\gamma_t^2}{2}\mathbb{E}\left\|\sum_{k=1}^{K} \nabla f_{\mathcal{G}(k),x_{i(t-K+k)}}(w^{t-K+k}) - \nabla f(w^t)\right\|^2}_{Q_1} \\
&\quad \underbrace{- (\gamma_t - L\gamma_t^2) \sum_{k=1}^{K} \nabla f(w^t)^T \left(\nabla f_{\mathcal{G}(k)}\left(w^{t-K+k}\right) - \nabla f_{\mathcal{G}(k)}\left(w^t\right)\right),}_{Q_2} \qquad (5.14)
\end{aligned}
$$

where the second inequality follows from the unbiased gradient $\mathbb{E}\left[\nabla f_{x_i}(w)\right] = \nabla f(w)$. Because of $\|x+y\|_2^2 \leq 2\|x\|_2^2 + 2\|y\|_2^2$ and $xy \leq \frac{1}{2}\|x\|_2^2 + \frac{1}{2}\|y\|_2^2$, we have the upper bound of Q_1 as follows:

$$
\begin{aligned}
Q_1 &= \frac{L\gamma_t^2}{2}\mathbb{E}\Big\| \sum_{k=1}^{K} \nabla f_{\mathcal{G}(k),x_{i(t-K+k)}}(w^{t-K+k}) - \nabla f(w^t) - \sum_{k=1}^{K} \nabla f_{\mathcal{G}(k)}(w^{t-K+k}) \\
&\quad + \sum_{k=1}^{K} \nabla f_{\mathcal{G}(k)}(w^{t-K+k}) \Big\|_2^2 \\
&\leq L\gamma_t^2 \underbrace{\mathbb{E}\left\| \sum_{k=1}^{K} \nabla f_{\mathcal{G}(k),x_{i(t-K+k)}}(w^{t-K+k}) - \sum_{k=1}^{K} \nabla f_{\mathcal{G}(k)}(w^{t-K+k}) \right\|_2^2}_{Q_3} \\
&\quad + L\gamma_t^2 \underbrace{\left\| \sum_{k=1}^{K} \nabla f_{\mathcal{G}(k)}(w^{t-K+k}) - \nabla f(w^t) \right\|_2^2}_{Q_4}. \qquad (5.15)
\end{aligned}
$$

We also have the upper bound of Q_2 as follows:

$$
\begin{aligned}
Q_2 &= -(\gamma_t - L\gamma_t^2) \sum_{k=1}^{K} \nabla f(w^t)^T \left(\nabla f_{\mathcal{G}(k)} \left(w^{t-K+k} \right) - \nabla f_{\mathcal{G}(k)} \left(w^t \right) \right) \\
&\leq \frac{\gamma_t - L\gamma_t^2}{2} \left\| \nabla f(w^t) \right\|_2^2 + \frac{\gamma_t - L\gamma_t^2}{2} \left\| \sum_{k=1}^{K} \nabla f_{\mathcal{G}(k)}(w^{t-K+k}) - \nabla f(w^t) \right\|_2^2. \qquad (5.16)
\end{aligned}
$$

As per the equation regarding variance $\mathbb{E}\|\xi - \mathbb{E}[\xi]\|_2^2 = \mathbb{E}\|\xi\|_2^2 - \|\mathbb{E}[\xi]\|_2^2$, we can bound Q_3 as follows:

$$
\begin{aligned}
Q_3 &= \mathbb{E}\left\| \sum_{k=1}^{K} \nabla f_{\mathcal{G}(k),x_{i(t-K+k)}}(w^{t-K+k}) - \sum_{k=1}^{K} \nabla f_{\mathcal{G}(k)}(w^{t-K+k}) \right\|_2^2 \\
&= \sum_{k=1}^{K} \mathbb{E}\left\| \nabla f_{\mathcal{G}(k),x_{i(t-K+k)}}(w^{t-K+k}) - \nabla f_{\mathcal{G}(k)}(w^{t-K+k}) \right\|_2^2 \\
&\leq \sum_{k=1}^{K} \mathbb{E}\left\| \nabla f_{\mathcal{G}(k),x_{i(t-K+k)}}(w^{t-K+k}) \right\|_2^2 \\
&\leq KM, \qquad (5.17)
\end{aligned}
$$

where the equality follows from the definition of $\nabla f_{\mathcal{G}(k)}(w)$ such that $[\nabla f_{\mathcal{G}(k)}(w)]_j = 0,\ \forall j \notin \mathcal{G}(k)$ and the last inequality is from Assumption 5.4.2. We can also get the upper bound of Q_4:

$$\begin{aligned}
Q_4 &= \left\| \sum_{k=1}^{K} \nabla f_{\mathcal{G}(k)}(w^{t-K+k}) - \nabla f(w^t) \right\|_2^2 \\
&= \sum_{k=1}^{K} \left\| \nabla f_{\mathcal{G}(k)}(w^{t-K+k}) - \nabla f_{\mathcal{G}(k)}(w^t) \right\|_2^2 \\
&\leq \sum_{k=1}^{K} \left\| \nabla f(w^{t-K+k}) - \nabla f(w^t) \right\|_2^2 \\
&\leq L^2 \sum_{k=1}^{K} \left\| \sum_{j=\max\{0,t-K+k\}}^{t-1} \left(w^{j+1} - w^j\right) \right\|^2 \\
&\leq L^2 \gamma^2_{\max\{0,t-K+1\}} K \sum_{k=1}^{K} \sum_{j=\max\{0,t-K+k\}}^{t-1} \left\| \sum_{k=1}^{K} \nabla f_{\mathcal{G}(k),x_{(j)}} \left(w^{j-K+k}\right) \right\|_2^2 \\
&\leq K L \gamma_t \frac{\gamma_{\max\{0,t-K+1\}}}{\gamma_t} \sum_{k=1}^{K} \sum_{j=\max\{0,t-K+k\}}^{t-1} \left\| \sum_{k=1}^{K} \nabla f_{\mathcal{G}(k),x_{(j)}} \left(w^{j-K+k}\right) \right\|_2^2 \\
&\leq L \gamma_t \sigma K^4 M, \qquad (5.18)
\end{aligned}$$

where the second inequality is from Assumption 5.4.1, the fourth inequality follows from that $L\gamma_t \leq 1$ and the last inequality follows from $\|z_1 + ... + z_r\|_2^2 \leq r(\|z_1\|_2^2 + ... + \|z_r\|_2^2)$, Assumption 5.4.2 and $\sigma := \max_t \frac{\gamma_{\max\{0,t-K+1\}}}{\gamma_t}$. Integrating the upper bound of Q_1, Q_2, Q_3 and Q_4 in (5.14), we have:

$$\begin{aligned}
\mathbb{E}\left[f(w^{t+1})\right] - f(w^t) &\leq -\frac{\gamma_t}{2} \left\| \nabla f(w^t) \right\|_2^2 + \gamma_t^2 L \sum_{k=1}^{K} \mathbb{E} \left\| \nabla f_{\mathcal{G}(k),x_{i(t-K+k)}}(w^{t-K+k}) \right\|^2 \\
&\quad + \frac{\gamma_t + L\gamma_t^2}{2} \left\| \sum_{k=1}^{K} \nabla f_{\mathcal{G}(k)}(w^{t-K+k}) - \nabla f(w^t) \right\|_2^2. \\
&\leq -\frac{\gamma_t}{2} \left\| \nabla f(w^t) \right\|^2 + \gamma_t^2 L M_K, \qquad (5.19)
\end{aligned}$$

where we let $M_K = KM + \sigma K^4 M$. □

From Lemma 5.4.1, we can observe that the expected decrease of the objective function is controlled by the learning rate γ_t and M_K. Therefore, we can guarantee that the values of objective functions are decreasing as long as the learning rates γ_t are small enough such that the right-hand side of (5.12) is less than zero. Using the lemma above, we can analyze the convergence property for Algorithm 9.

5.4.1 Fixed Learning Rate

Firstly, we analyze the convergence for Algorithm 9 when γ_t is fixed and prove that the learned model will converge sub-linearly to the neighborhood of the critical points.

Theorem 5.4.1. *Assume Assumption 5.4.1 and 5.4.2 hold and the fixed learning rate sequence $\{\gamma_t\}$ satisfies $\gamma_t = \gamma$ and $\gamma L \leq 1, \forall t \in \{0, 1, ..., T-1\}$. In addition, we assume w^* to be the optimal solution to $f(w)$ and let $\sigma = 1$ such that $M_K = KM + K^4M$. Then, the output of Algorithm 9 satisfies that:*

$$\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\left\|\nabla f(w^t)\right\|_2^2 \leq \frac{2\left(f(w^0) - f(w^*)\right)}{\gamma T} + 2\gamma L M_K \tag{5.20}$$

Proof of Theorem 5.4.1: When γ_t is constant and $\gamma_t = \gamma$, taking total expectation of (5.12) in Lemma 5.4.1, we obtain:

$$\mathbb{E}\left[f(w^{t+1})\right] - \mathbb{E}\left[f(w^t)\right] \leq -\frac{\gamma}{2}\mathbb{E}\left\|\nabla f(w^t)\right\|_2^2 + \gamma^2 L M_K, \tag{5.21}$$

where $\sigma = 1$ and $M_K = KM + K^4M$. Summing (5.21) from $t = 0$ to $T-1$, we have:

$$\mathbb{E}\left[f(w^T)\right] - f(w^0) \;\; \leq \;\; -\frac{\gamma}{2}\sum_{t=0}^{T-1}\mathbb{E}\left\|\nabla f(w^t)\right\|_2^2 + T\gamma^2 L M_K. \tag{5.22}$$

Suppose w^* is the optimal solution for $f(w)$, therefore $f(w^*) - f(w^0) \leq \mathbb{E}\left[f(w^T)\right] - f(w^0)$. Above all, the following inequality is guaranteed that:

$$\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\left\|\nabla f(w^t)\right\|_2^2 \leq \frac{2\left(f(w^0) - f(w^*)\right)}{\gamma T} + 2\gamma L M_K. \tag{5.23}$$

□

In Theorem 5.4.1, we can observe that when $T \to \infty$, the average norm of the gradients is upper bounded by $2\gamma L M_K$. The number of modules K affects the value of the upper bound. Selecting a small learning rate γ allows us to get better neighborhood to the critical points, however it also seriously decreases the speed of convergence.

5.4.2 Diminishing Learning Rate

In this section, we prove that Algorithm 9 with diminishing learning rates can guarantee the convergence to critical points for the non-convex problem.

Theorem 5.4.2. *Assume Assumption 5.4.1 and 5.4.2 hold and the diminishing learning rate sequence* $\{\gamma_t\}$ *satisfies* $\gamma_t = \frac{\gamma_0}{1+t}$ *and* $\gamma_t L \leq 1, \forall t \in \{0, 1, ..., T-1\}$. *In addition, we assume* w^* *to be the optimal solution to* $f(w)$ *and let* $\sigma = K$ *such that* $M_K = KM + K^5 M$. *Setting* $\Gamma_T = \sum_{t=0}^{T-1} \gamma_t$, *then the output of Algorithm 9 satisfies that:*

$$\frac{1}{\Gamma_T} \sum_{t=0}^{T-1} \gamma_t \mathbb{E} \left\| \nabla f(w^t) \right\|_2^2 \quad \leq \quad \frac{2\left(f(w^0) - f(w^*)\right)}{\Gamma_T} + \frac{2\sum_{t=0}^{T-1} \gamma_t^2 L M_K}{\Gamma_T}. \tag{5.24}$$

Proof of Theorem 5.4.2: $\{\gamma_t\}$ is a diminishing sequence and $\gamma_t = \frac{\gamma_0}{1+t}$, such that $\sigma \leq K$ and $M_K = KM + K^5 M$. Taking total expectation of (5.12) in Lemma 5.4.1 and summing it from $t = 0$ to $T - 1$, we obtain:

$$\mathbb{E}\left[f(w^T)\right] - f(w^0) \quad \leq \quad -\frac{1}{2} \sum_{t=0}^{T-1} \gamma_t \mathbb{E} \left\| \nabla f(w^t) \right\|_2^2 + \sum_{t=0}^{T-1} \gamma_t^2 L M_K. \tag{5.25}$$

Suppose w^* is the optimal solution for $f(w)$, therefore $f(w^*) - f(w^0) \leq \mathbb{E}\left[f(w^T)\right] - f(w^0)$. Letting $\Gamma_T = \sum_{t=0}^{T-1} \gamma_t$, we have:

$$\frac{1}{\Gamma_T} \sum_{t=0}^{T-1} \gamma_t \mathbb{E} \left\| \nabla f(w^t) \right\|_2^2 \leq \frac{2\left(f(w^0) - f(w^*)\right)}{\Gamma_T} + \frac{2\sum_{t=0}^{T-1} \gamma_t^2 L M_K}{\Gamma_T}. \tag{5.26}$$

We complete the proof. □

Table 4: Neural networks architectural details in the experiments.

Architecture	Units	Channels
ResNet-8	1-1-1	16-16-32-64
ResNet-56	9-9-9	16-16-32-64
ResNet-110	18-18-18	16-16-32-64

Corollary 5.4.1. *Since $\gamma_t = \frac{\gamma_0}{t+1}$, the learning rate requirements in [84] are satisfied that:*

$$\lim_{T\to\infty} \sum_{t=0}^{T-1} \gamma_t = \infty \quad \textit{and} \quad \lim_{T\to\infty} \sum_{t=0}^{T-1} \gamma_t^2 < \infty. \tag{5.27}$$

Therefore, according to Theorem 5.4.2, when $T \to \infty$, the right-hand side of (5.24) converges to 0.

Corollary 5.4.2. *Suppose w^s is chosen randomly from $\{w^t\}_{t=0}^{T-1}$ with probabilities proportional to $\{\gamma_t\}_{t=0}^{T-1}$. According to Theorem 5.4.2, we can prove that Algorithm 9 guarantees convergence to critical points for the non-convex problem:*

$$\lim_{s\to\infty} \mathbb{E}\|\nabla f(w^s)\|_2^2 \;=\; 0. \tag{5.28}$$

5.5 Experimental Results

In this section, we experiment with ResNet [30] on image classification benchmark datasets: CIFAR-10 and CIFAR-100 [53]. In section 5.5.1, we evaluate our method by varying the positions and the number of the split points in the network; In section 5.5.2 we use our method to optimize deeper neural networks and show that its performance is as good as the performance of backpropagation; finally, we split and distribute the ResNet-110

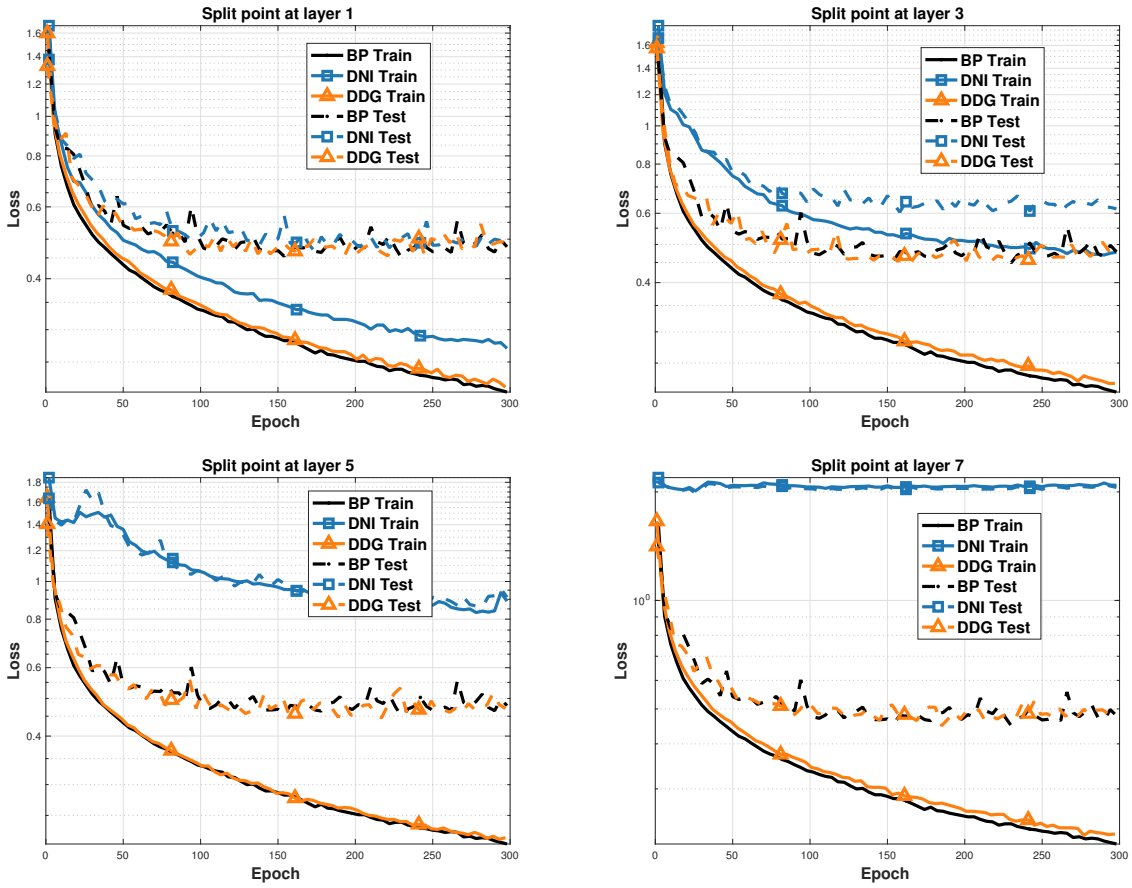


Figure 17: Training and testing curves of loss function regarding epochs for ResNet-8 on CIFAR-10.

across GPUs in Section 5.5.3, results showing that the proposed method achieves a speedup of two times without loss of accuracy.

Implementation Details: We implement DDG algorithm using PyTorch library [77]. The trained network is split into K modules where each module is running on a subprocess. The subprocesses are spawned using multiprocessing package [1] such that we can fully leverage multiple processors on a given machine. Running modules on different subprocesses make the communication very difficult. To make the communication fast, we utilize the shared memory objects in the multiprocessing package.

[1] https://docs.python.org/3/library/multiprocessing.html#module-multiprocessing

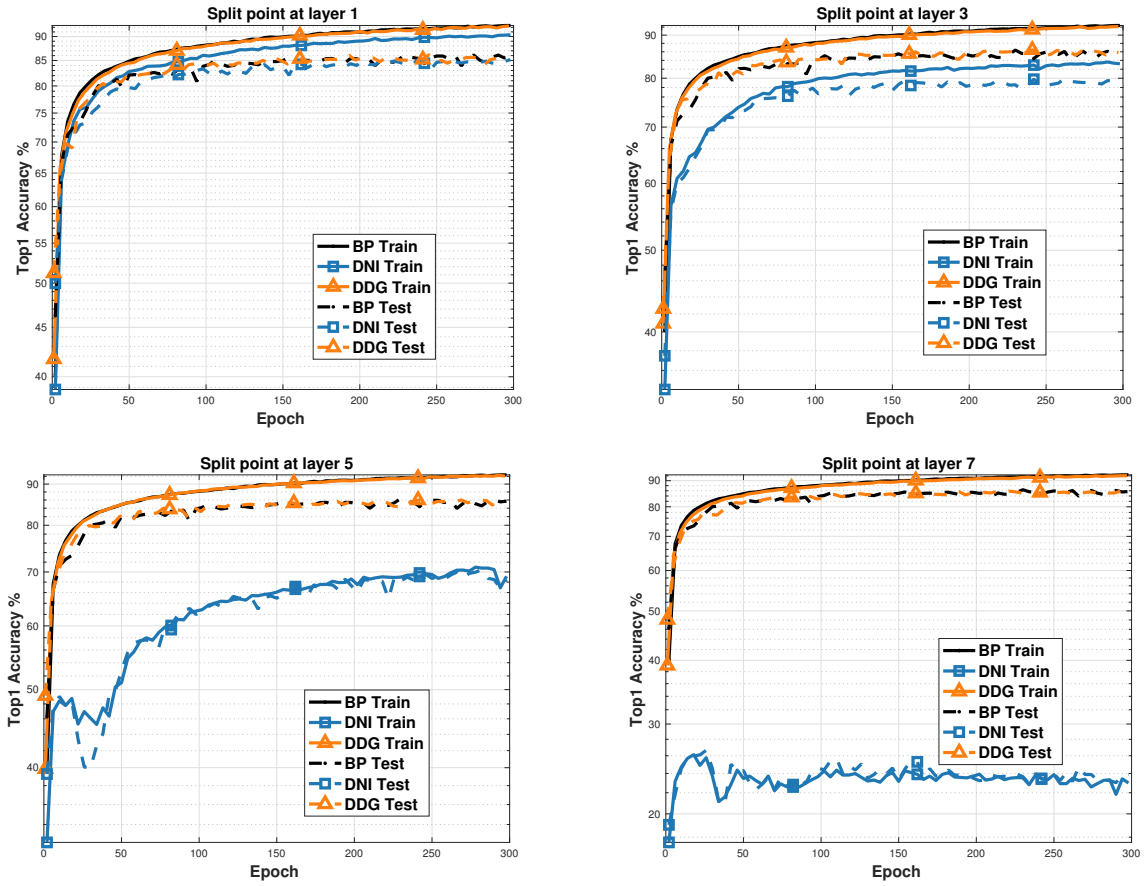


Figure 18: Training and testing curves of Top 1 classification accuracies regarding epochs for ResNet-8 on CIFAR-10.

5.5.1 Comparison of BP, DNI and DDG

In this section, we train ResNet-8 on CIFAR-10 on a single Titan X GPU. The architecture of the ResNet-8 is in Table 4. All experiments are run for 300 epochs and optimized using Adam optimizer [50] with a batch size of 128. The learning rate is initialized at 1×10^{-3}. We augment the dataset with random cropping, random horizontal flipping and normalize the image using mean and standard deviation. There are three compared methods in this experiment:

- **BP**: Adam in Pytorch uses backpropagation algorithm [85] to compute gradients.

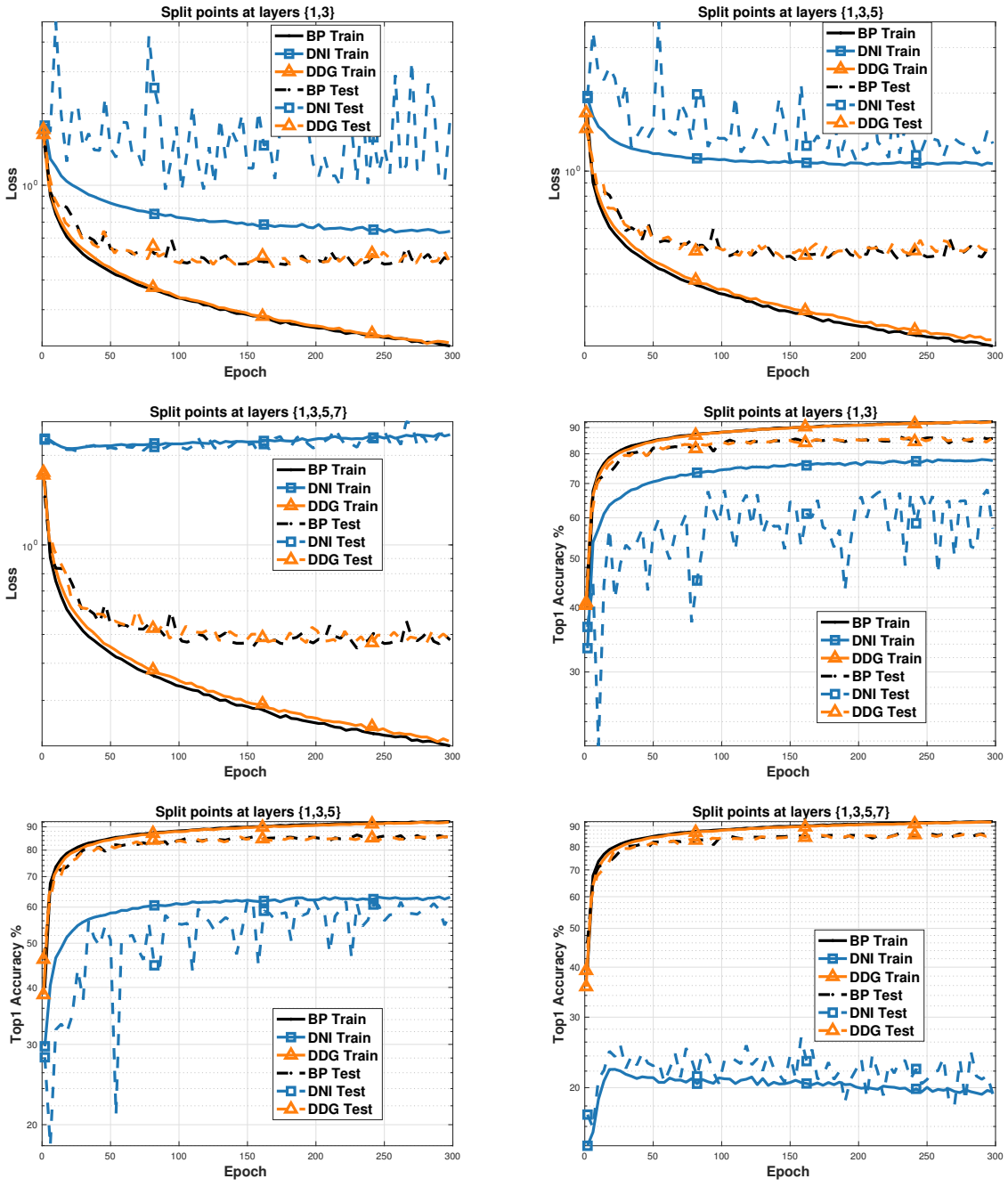
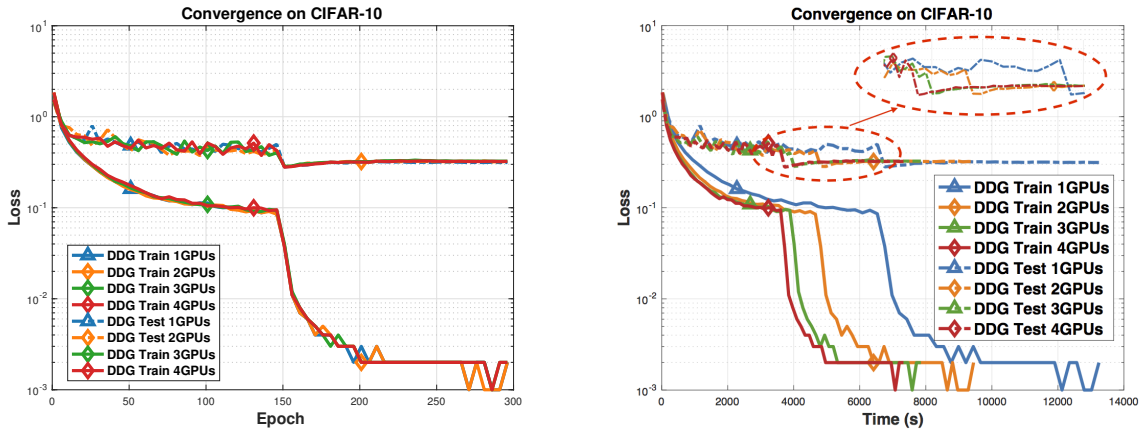


Figure 19: Training and testing curves regarding epochs for ResNet-8 on CIFAR-10.



(a) (b)

Figure 20: Training and testing loss curves for ResNet-110 on CIFAR-10 using multiple GPUs.

- **DNI**: Decoupled neural interface (DNI) in [42]. Following [42], the synthetic network is a stack of three convolutional layers with L 5×5 filters with resolution preserving padding. The filter depth L is determined by the position of DNI. We also input label information into the synthetic network to increase final accuracy.
- **DDG**: Adam optimizer using delayed gradients in Algorithm 10.

Impact of split position (depth). The position (depth) of the split points determines the number of layers using delayed gradients. Stale or synthetic gradients will induce noises in the training process, affecting the convergence of the objective. Figure2 17 and 18 exhibit the experimental results when there is only one split point with varying positions. In the first column, we know that all compared methods have similar performances when the split point is at layer 1. DDG performs consistently well when we place the split point at deeper positions $3, 5$ or 7. On the contrary, the performance of DNI degrades as we vary the positions and it cannot even converge when the split point is at layer 7.

Impact of the number of split points. From equation (5.7), we know that the maximum time delay is determined by the number of modules K. Theorem 5.4.2 also shows

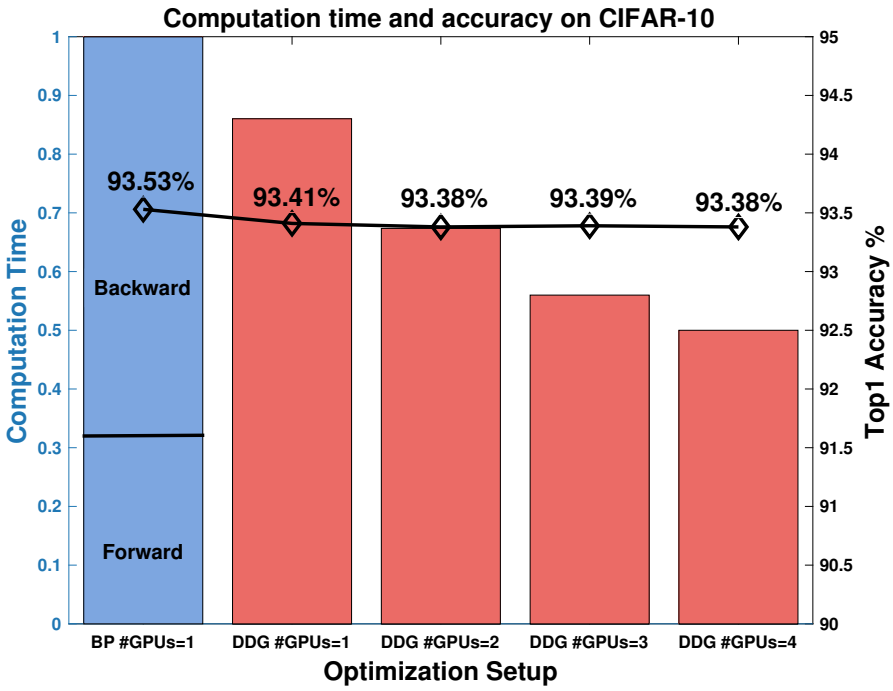


Figure 21: Computation time and the best Top 1 accuracy for ResNet-110 on the test data of CIFAR-10.

that K affects the convergence rate. In this experiment, we vary the number of split points in the network from 2 to 4 and plot the results in Figure 19. It is easy to observe that DDG performs as well as BP, regardless of the number of split points in the network. However, DNI is very unstable when we place more split points, and cannot even converge sometimes.

5.5.2 Optimizing Deeper Neural Networks

In this section, we employ DDG to optimize two very deep neural networks (ResNet-56 and ResNet-110) on CIFAR-10 and CIFAR-100. Each network is split into two modules at the center. We use SGD with the momentum of 0.9 and the learning rate is initialized to 0.01. Each model is trained for 300 epochs and the learning rate is divided by a factor of 10 at 150 and 225 epochs. The weight decay constant is set to 5×10^{-4}. We perform the same

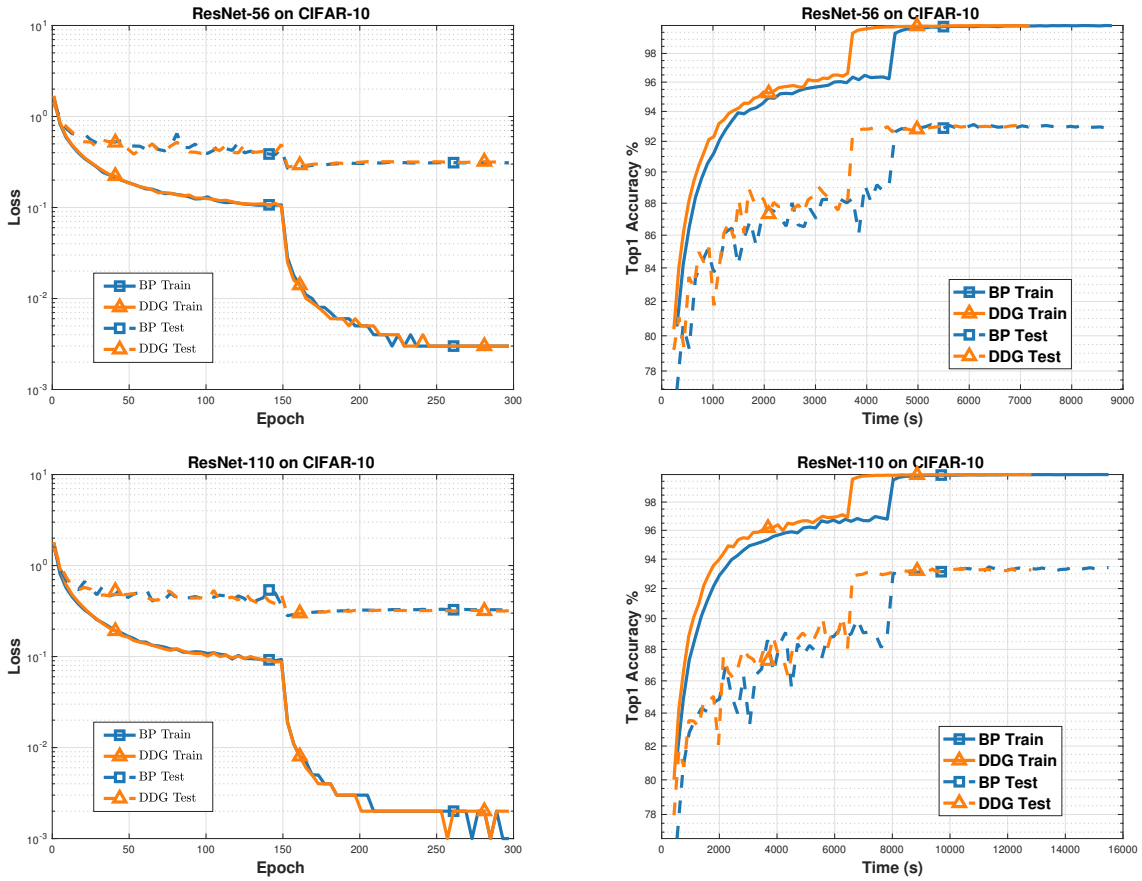


Figure 22: Training and testing curves for ResNet-56 and ResNet-110 on CIFAR-10.

data augmentation as in section 5.5.1. Experiments in this section are run on a single Titan X GPU.

Figures 22 and 23 presents the experimental results of BP and DDG. We do not compare DNI because its performance is far worse when models are deep. Figures in the first column present the convergence of loss regarding epochs, showing that DDG and BP admit similar convergence rates. We can also observe that DDG converges faster when we compare the accuarcy regarding computation time in Figures 22 and 23. In the experiment, the "Volatile GPU Utility" is about 70% when we train the models with BP. Our method runs on two subprocesses such that it fully leverages the computing capacity of the GPU. In Table 5, we list the best Top 1 accuracy on the test data of CIFAR-10 and CIFAR-100. We can observe

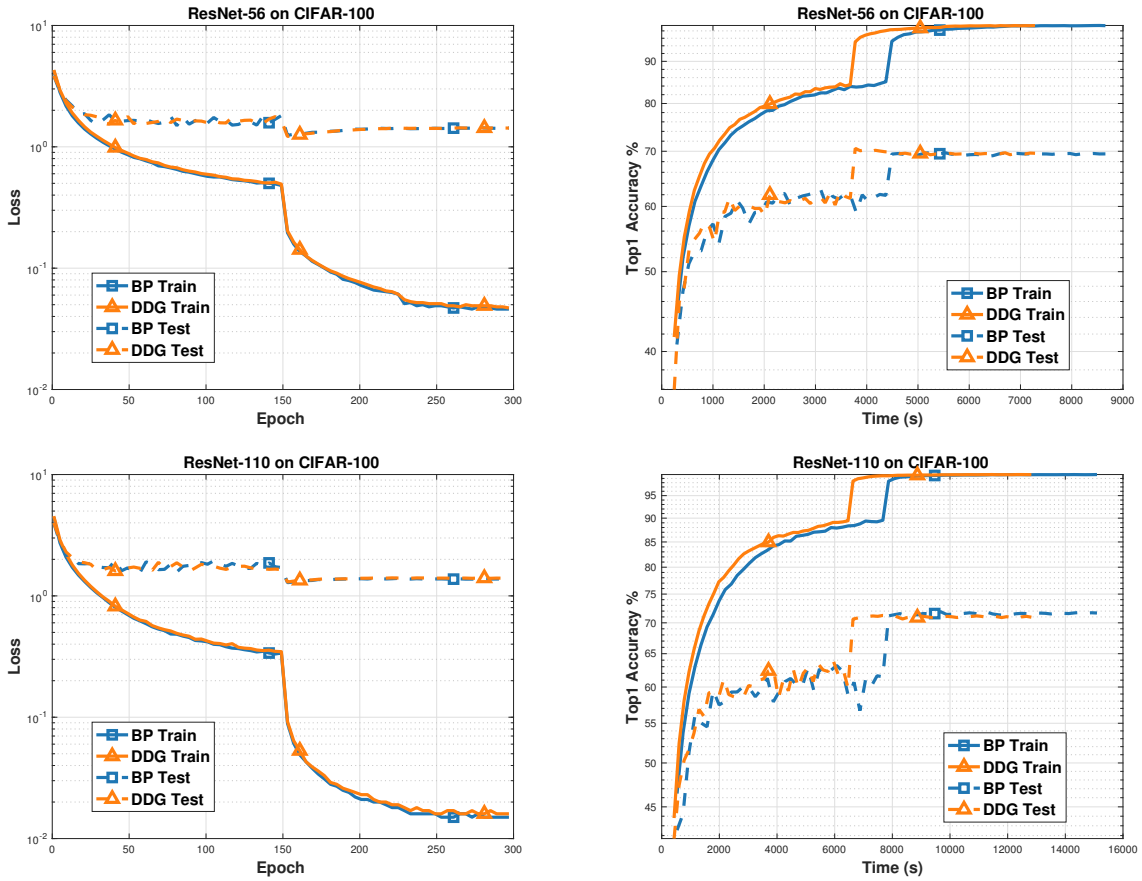


Figure 23: Training and testing curves for ResNet-56 and ResNet-110 on CIFAR-100.

that the prorposed DDG algorithm can obtain comparable or better accuracy even when the network is deep.

5.5.3 Scaling the Number of GPUs

In this section, we split ResNet-110 into K modules and allocate them across K Titan X GPUs sequentially. We do not consider filter-wise model parallelism in this experiment. The selections of the parameters in the experiment are similar to Section 5.5.2. From Figure 20, we know that training networks in multiple GPUs does not affect the convergence rate. For comparison, we also count the computation time of backpropagation algorithm on a

Table 5: The best Top 1 classification accuracy for ResNet-56 and ResNet-110 on the test data of CIFAR-10 and CIFAR-100.

Architecture	CIFAR-10		CIFAR-100	
	BP	DDG	BP	DDG
ResNet-56	**93.12**	93.11	69.79	**70.17**
ResNet-110	**93.53**	93.41	**71.90**	71.39

single GPU. The computation time is worse when we run backpropagation algorithm on multiple GPUs because of the communication overhead. In Figure 21, we can observe that forward time only accounts for about 32% of the total computation time for backpropagation algorithm. Therefore, backward locking is the main bottleneck. In Figure 21, it is obvious that when we increase the number of GPUs from 2 to 4, our method reduces about 30% to 50% of the total computation time. In other words, DDG achieves a speedup of about 2 times without loss of accuracy when we train the networks across 4 GPUs.

6.0 Training Neural Networks Using Features Replay

6.1 Motivation

In recent years, the deep convolutional neural networks have made great breakthroughs in computer vision [30, 37, 54, 56, 95, 96], natural language processing [47, 49, 87, 110], and reinforcement learning [61, 69, 70, 71]. The growth of the depths of the neural networks is one of the most critical factors contributing to the success of deep learning, which has been verified both in practice [30, 37] and in theory [8, 21, 99]. Gradient-based methods are the major methods to train deep neural networks, such as stochastic gradient descent (SGD) [84], ADAGRAD [20], RMSPROP [32] and ADAM [50]. As long as the loss functions are differentiable, we can compute the gradients of the networks using backpropagation algorithm [85]. The backpropagation algorithm requires two passes of the neural network, the forward pass to compute activations and the backward pass to compute gradients. As shown in Figure 24 (BP), error gradients are repeatedly propagated from the top (output layer) all the way back to the bottom (input layer) in the backward pass. The sequential propagation of the error gradients is called backward locking because all layers of the network are locked until their dependencies have executed. According to the benchmark report in [45], the computational time of the backward pass is about twice of the computational time of the forward pass. When networks are quite deep, backward locking becomes the bottleneck of making good use of computing resources, preventing us from updating layers in parallel.

There are several works trying to break the backward locking in the backpropagation algorithm. [11] and [98] avoid the backward locking by removing the backpropagation algorithm completely. In [11], the authors proposed the method of auxiliary coordinates (MAC) and simplified the nested functions by imposing quadratic penalties. Similarly, [98] used Lagrange multipliers to enforce equality constraints between auxiliary variables and activations. Both of the reformulated problems do not require backpropagation algorithm at all and are easy to be parallelized. However, neither of them have been applied to training convolutional neural networks yet. There are also several works breaking the dependencies

between groups of layers or modules in the backpropagation algorithm. In [42], the authors proposed to remove the backward locking by employing the decoupled neural interface to approximate error gradients (Figure 24 DNI). [6, 76] broke the local dependencies between successive layers and made all hidden layers receive error information from the output layer directly. In the backward pass, we can use the synthetic gradients or the direct feedbacks to update the weights of all modules without incurring any delay. However, these methods work poorly when the neural networks use very deep architecture. In [38], the authors proposed decoupled parallel backpropagation by using stale gradients, where modules are updated with the gradients from different timestamps (Figure 24 DDG). However, it requires large amounts of memory to store the stale gradients and suffers from the loss of accuracy.

In this chapter, we propose feature replay algorithm which is free of the above three issues: backward locking, memory explosion and accuracy loss. The main contributions of our work are summarized as follows:

- Firstly, we propose a novel parallel-objective formulation for the objective function of the neural networks in Section 5.3. Using this new formulation, we break the backward locking by introducing features replay algorithm, which is easy to be parallelized.
- Secondly, we provide the theoretical analysis in Section 5.4 and prove that the proposed method is guaranteed to converge to critical points for the non-convex problem under certain conditions.
- Finally, we validate our method with experiments on training deep convolutional neural networks in Section 5.5. Experimental results demonstrate that the proposed method achieves faster convergence, lower memory consumption, and better generalization error than compared methods.

6.2 Preliminaries

We assume a feedforward neural network with L layers, where $w = [w_1, w_2, ..., w_L] \in \mathbb{R}^d$ denotes the weights of all layers. The computation in each layer can be represented as taking

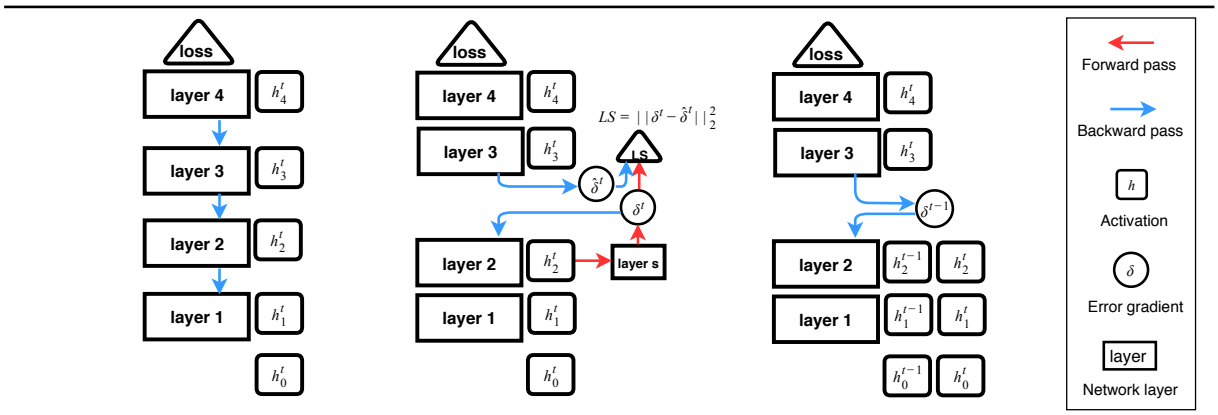


Figure 24: Illustrations of the backward pass of the backpropagation algorithm (BP) decoupled neural interface (DNI) and decoupled parallel backpropagation (DDG).

an input h_{l-1} and producing an activation $h_l = F_l(h_{l-1}; w_l)$ using weight w_l. Given a loss function f and target y, we can formulate the objective function of the neural network $f(w)$ as follows:

$$\begin{aligned} \min_{w} \quad & f(h_L, y) \\ s.t. \quad & h_l = F_l(h_{l-1}; w_l) \quad \text{for all} \quad l \in \{1, 2, ..., L\}. \end{aligned} \tag{6.1}$$

where h_0 denotes the input data x. By using stochastic gradient descent, the weights of the network are updated in the direction of their negative gradients of the loss function following:

$$w_l^{t+1} \quad = \quad w_l^t - \gamma_t \cdot g_l^t \quad \text{for all} \quad l \in \{1, 2, ..., L\}. \tag{6.2}$$

where γ_t denotes the learning rate and $g_l^t := \frac{\partial f_{x^t}(w^t)}{\partial w_l^t}$ denotes the gradient of the loss function (6.1) regarding w_l^t with input samples x^t. The backpropagation algorithm [85] is utilized to compute the gradients for the neural networks. At iteration t, it requires two passes over the network: in the forward pass, the activations of all layers are computed from the bottom layer $l = 1$ to the top layer $l = L$ following: $h_l^t = F_l(h_{l-1}^t; w_l^t)$; in the backward pass, it

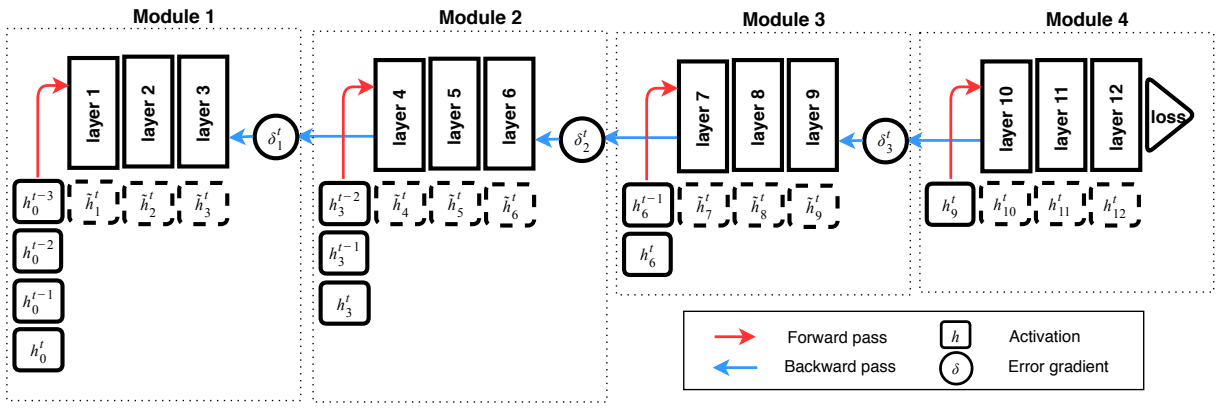


Figure 25: Backward pass of Features Replay Algorithm.

applies the chain rule and propagates error gradients through the network from the top layer $l = L$ to the bottom layer $l = 1$ following:

$$\frac{\partial f_{x^t}(w^t)}{\partial w_l^t} = \frac{\partial h_l^t}{\partial w_l^t} \times \frac{\partial f_{x^t}(w^t)}{\partial h_l^t} \tag{6.3}$$

$$\frac{\partial f_{x^t}(w^t)}{\partial h_{l-1}^t} = \frac{\partial h_l^t}{\partial h_{l-1}^t} \times \frac{\partial f_{x^t}(w^t)}{\partial h_l^t}. \tag{6.4}$$

According to (6.4), computing gradients for the weights w^l of the layer l is dependent on the error gradient $\frac{\partial f_{x^t}(w^t)}{\partial h_l^t}$ from the layer $l+1$, which is known as backward locking. Therefore, the backward locking prevents all layers from updating before receiving error gradients from dependent layers. When the networks are deep, the backward locking becomes the bottleneck in the training process.

6.3 Features Replay

In this section, we propose a novel parallel-objective formulation for the objective function of the neural networks. Using our new formulation, we break the backward locking in the backpropagation algorithm by using features replay algorithm.

6.3.1 Problem Reformulation

As shown in Figure 25, we assume to divide an L-layer feedforward neural network into K modules where $K \ll L$, such that $w = [w_{\mathcal{G}(1)}, w_{\mathcal{G}(2)}, ..., w_{\mathcal{G}(K)}] \in \mathbb{R}^d$ and $\mathcal{G}(k)$ denotes the layers in the module k. Let L_k represent the last layer of the module k, the output of this module can be written as h_{L_k}. The error gradient variable is denoted as δ_k^t , which is used for the gradient computation of the module k. We divide a 12-layer neural network into four modules, where each module stores its input history and a stale error gradient from the upper module. At each iteration, all modules compute the activations by inputting features from the history and compute the gradients by applying the chain rule. After that, they receive the error gradients from the upper modules for the next iteration. We can split the problem (6.1) into K subproblems. The task of the module k (except $k = K$) is minimizing the least square error between the error gradient variable δ_k^t and $\frac{\partial f_{h_{L_k}^t}(w^t)}{\partial h_{L_k}^t}$ which is the gradient of the loss function regarding $h_{L_k}^t$ with input $h_{L_k}^t$ into the module $k+1$, and the task of the module K is minimizing the loss between the prediction $h_{L_K}^t$ and the real label y^t. From this point of view, we propose a novel parallel-objective loss function at iteration t as follows:

$$\begin{aligned} \min_{w,\delta} \quad & \sum_{k=1}^{K-1} \left\| \delta_k^t - \frac{\partial f_{h_{L_k}^t}(w^t)}{\partial h_{L_k}^t} \right\|_2^2 + f\left(h_{L_K}^t, y^t\right) \\ s.t. \quad & h_{L_k}^t = F_{\mathcal{G}(k)}(h_{L_{k-1}}^t; w_{\mathcal{G}(k)}^t) \quad \text{for all} \quad k \in \{1, ..., K\}, \end{aligned} \tag{6.5}$$

where $h_{L_0}^t$ denotes the input data x^t. It is obvious that the optimal solution for the left term of the problem (6.5) is $\delta_k^t = \frac{\partial f_{h_{L_k}^t}(w^t)}{\partial h_{L_k}^t}$, for all $k \in \{1, ..., K-1\}$. In other words, the optimal solution of the module k is dependent on the output of the upper modules. Therefore, minimizing the problem (6.1) with the backpropagation algorithm is equivalent to minimizing the problem (6.5) with the first $K-1$ subproblems obtaining optimal solutions.

6.3.2 Breaking Dependencies by Replaying Features

Features replay algorithm is introduced in Algorithm 11. In the forward pass, immediate features are generated and passed through the network, and the module k keeps a history of its input with size $K - k + 1$. To break the dependencies between modules in the backward

Algorithm 11 Features Replay Algorithm

Initialize: weights $w^0 = [w^0_{\mathcal{G}(1)}, ..., w^0_{\mathcal{G}(K)}] \in \mathbb{R}^d$ and learning rate sequence $\{\gamma_t\}$;
for $t = 0, 1, 2, \ldots, T-1$ **do**
  Sample mini-batch (x^t, y^t) from the dataset and let $h^t_{L_0} = x^t$;
  for $k = 1, \ldots, K$ **do**
    Store $h^t_{L_{k-1}}$ in the memory;
    Compute $h^t_{L_k}$ following $h^t_{L_k} = F_{\mathcal{G}(k)}\left(h^t_{L_{k-1}}; w^t_{\mathcal{G}(k)}\right)$; $\leftarrow$ Play
    Send $h^t_{L_k}$ to the module $k+1$ if $k < K$;
  end for
  Compute loss $f(w^t) = f\left(h^t_{L_K}, y^t\right)$;
  for $k = 1, \ldots, K$ **in parallel do**
    Compute $\tilde{h}^t_{L_k}$ following $\tilde{h}^t_{L_k} = F_{\mathcal{G}(k)}(h^{t+k-K}_{L_{k-1}}; w^t_{\mathcal{G}(k)})$; $\leftarrow$ Replay
    Compute gradient $g^t_{\mathcal{G}(k)}$ following (6.9);
    Update weights: $w^{t+1}_{\mathcal{G}(k)} = w^t_{\mathcal{G}(k)} - \gamma_t \cdot g^t_{\mathcal{G}(k)}$;
    Send $\frac{\partial f_{h^{t+k-K}_{L_{k-1}}}(w^t)}{\partial h^{t+k-K}_{L_{k-1}}}$ to the module $k-1$ if $k > 1$;
  end for
end for

Forward pass

Backward pass

pass, we propose to compute the gradients of the modules using immediate features from different timestamps. *Features replay* denotes that immediate feature $h^{t+k-K}_{L_{k-1}}$ is input into the module k for the first time in the forward pass at iteration $t+k-K$, and it is input into the module k for the second time in the backward pass at iteration t. If $t+k-K < 0$, we set $h^{t+k-K}_{L_{k-1}} = 0$. Therefore, the new problem can be written as:

$$\begin{aligned} \min_{w,\delta} \quad & \sum_{k=1}^{K-1} \left\| \delta^t_k - \frac{\partial f_{\tilde{h}^t_{L_k}}(w^t)}{\partial \tilde{h}^t_{L_k}} \right\|_2^2 + f(\tilde{h}^t_{L_K}, y^t) \\ s.t. \quad & \tilde{h}^t_{L_k} = F_{\mathcal{G}(k)}(h^{t+k-K}_{L_{k-1}}; w^t_{\mathcal{G}(k)}) \quad \text{for all} \quad k \in \{1, ..., K\}. \end{aligned} \tag{6.6}$$

where $\frac{\partial f_{\tilde{h}^t_{L_k}}(w^t)}{\partial \tilde{h}^t_{L_k}}$ denotes the gradient of the loss $f(w^t)$ regarding $\tilde{h}^t_{L_k}$ with input $\tilde{h}^t_{L_k}$ into the module $k+1$. It is important to note that it is not necessary to get the optimal solutions for the first $K-1$ subproblems while we do not compute the optimal solution for the last

subproblem. To avoid the tedious computation, we make a trade-off between the error of the left term in (6.6) and the computational time by making:

$$\delta_k^t = \frac{\partial f_{h_{L_k}^{t+k-K}}(w^{t-1})}{\partial h_{L_k}^{t+k-K}} \quad \text{for all} \quad k \in \{1, ..., K-1\}, \tag{6.7}$$

where $\frac{\partial f_{h_{L_k}^{t+k-K}}(w^{t-1})}{\partial h_{L_k}^{t+k-K}}$ denotes the gradient of the loss $f(w^{t-1})$ regarding $h_{L_k}^{t+k-K}$ with input $h_{L_k}^{t+k-K}$ into the module $k+1$ at the previous iteration. Assuming the algorithm has converged as $t \to \infty$, we have $w^t \approx w^{t-1} \approx w^{t+k-K}$ such that $\tilde{h}_{L_k}^t \approx h_{L_k}^{t+k-K}$ and $\left\| \frac{\partial f_{h_{L_k}^{t+k-K}}(w^{t-1})}{\partial h_{L_k}^{t+k-K}} - \frac{\partial f_{\tilde{h}_{L_k}^t}(w^t)}{\partial \tilde{h}_{L_k}^t} \right\|_2^2 \approx 0$ for all $k \in \{1, ..., K-1\}$. Therefore, (6.7) is a reasonable approximation of the optimal solutions to the first $K-1$ subproblems in (6.6). In this way, we break the backward locking in the backpropagation algorithm because the error gradient variable δ_k^t can be determined at the previous iteration $t-1$ such that all modules are independent of each other at iteration t. Additionally, we compute the gradients inside each module following:

$$\frac{\partial f_{h_{L_{k-1}}^{t+k-K}}(w^t)}{\partial w_l^t} = \frac{\partial \tilde{h}_{L_k}^t}{\partial w_l^t} \times \delta_k^t \tag{6.8}$$

$$\frac{\partial f_{h_{L_{k-1}}^{t+k-K}}(w^t)}{\partial \tilde{h}_l^t} = \frac{\partial \tilde{h}_{L_k}^t}{\partial \tilde{h}_l^t} \times \delta_k^t, \tag{6.9}$$

where $l \in \mathcal{G}(k)$. At the end of each iteration, the module k sends $\frac{\partial f_{h_{L_{k-1}}^{t+k-K}}(w^t)}{\partial h_{L_{k-1}}^{t+k-K}}$ to module $k-1$ for the computation of the next iteration.

6.4 Convergence Analysis

In this section, we provide theoretical analysis for Algorithm 11. Analyzing the convergence of the problem (6.6) directly is difficult, as it involves the variables of different timestamps. Instead, we solve this problem by building a connection between the gradients of Algorithm 11 and stochastic gradient descent in Assumption 6.4.1, and prove that the proposed method is guaranteed to converge to critical points for the non-convex problem (6.1).

Assumption 6.4.1. ***(Sufficient direction)*** *We assume that the expectation of the descent direction* $\mathbb{E}\left[\sum_{k=1}^{K} g^t_{\mathcal{G}(k)}\right]$ *in Algorithm 11 is a sufficient descent direction of the loss* $f(w^t)$ *regarding* w^t*. Let* $\nabla f(w^t)$ *denote the full gradient of the loss, there exists a constant* $\sigma > 0$ *such that,*

$$\left\langle \nabla f(w^t), \mathbb{E}\left[\sum_{k=1}^{K} g^t_{\mathcal{G}(k)}\right] \right\rangle \;\geq\; \sigma \|\nabla f(w^t)\|_2^2. \tag{6.10}$$

Sufficient direction assumption guarantees that the model is moving towards the descending direction of the loss function.

Assumption 6.4.2. *Throughout this chapter, we make two assumptions following [10]:*

- ***(Lipschitz-continuous gradient)*** *The gradient of* f *is Lipschitz continuous with a constant* $L > 0$*, such that for any* $w_1, w_2 \in \mathbb{R}^d$*, it is satisfied that* $\|\nabla f(w_1) - \nabla f(w_2)\|_2 \leq L\|w_1 - w_2\|_2$.
- ***(Bounded variance)*** *We assume that the second moment of the descent direction in Algorithm 11 is upper bounded. There exists a constant* $M \geq 0$ *such that* $\mathbb{E}\left\|\sum_{k=1}^{K} g^t_{\mathcal{G}(k)}\right\|_2^2 \leq M$.

According to the equation regarding variance $\mathbb{E}\,\|\xi - \mathbb{E}\,[\xi]\|_2^2 = \mathbb{E}\,\|\xi\|_2^2 - \|\mathbb{E}\,[\xi]\|_2^2$, the variance of the descent direction $\mathbb{E}\left\|\sum_{k=1}^{K} g^t_{\mathcal{G}(k)} - \mathbb{E}\left[\sum_{k=1}^{K} g^t_{\mathcal{G}(k)}\right]\right\|_2^2$ is guaranteed to be less than M. According to the above assumptions, we prove the convergence rate for the proposed method under two circumstances of γ_t. Firstly, we analyze the convergence for Algorithm 11 when γ_t is fixed and prove that the learned model will converge sub-linearly to the neighborhood of the critical points for the non-convex problem.

Lemma 6.4.1. *Assume that Assumptions 6.4.1 and 6.4.2 hold. The iterations in Algorithm 11 satisfy the following inequality, for all* $t \in \mathbb{N}$*:*

$$\mathbb{E}[f(w^{t+1})] - f(w^t) \;\leq\; -\sigma\gamma_t \left\|\nabla f(w^t)\right\|_2^2 + \frac{\gamma_t^2 LM}{2}. \tag{6.11}$$

Proof of Lemma 6.4.1. Because the gradient of $f(w)$ is Lipschitz continuous in Assumption 6.4.2, the following inequality holds that:

$$f(w^{t+1}) \leq f(w^t) + \left\langle \nabla f(w^t), w^{t+1} - w^t \right\rangle + \frac{L}{2} \left\| w^{t+1} - w^t \right\|_2^2. \tag{6.12}$$

From the update rule in the Algorithm, we take expectation on both sides and obtain:

$$\begin{aligned} \mathbb{E}\left[f(w^{t+1})\right] &\leq f(w^t) - \gamma_t \sum_{k=1}^{K} \mathbb{E}\left\langle \nabla f(w^t), g_{\mathcal{G}(k)}^t \right\rangle + \frac{L\gamma_t^2}{2} \mathbb{E}\left\| \sum_{k=1}^{K} g_{\mathcal{G}(k)}^t \right\|_2^2 \\ &\leq f(w^t) - \sigma\gamma_t \left\| \nabla f(w^t) \right\|_2^2 + \frac{\gamma_t^2 LM}{2} \end{aligned} \tag{6.13}$$

where the second inequality follows from Assumptions 6.4.1 and 6.4.2. □

Theorem 6.4.1. *Assume that Assumptions 6.4.1 and 6.4.2 hold, and the fixed learning rate sequence $\{\gamma_t\}$ satisfies $\gamma_t = \gamma$ for all $t \in \{0, 1, ..., T-1\}$. In addition, we assume w^* to be the optimal solution to $f(w)$. Then, the output of Algorithm 11 satisfies that:*

$$\frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}\left\| \nabla f(w^t) \right\|_2^2 \leq \frac{f(w^0) - f(w^*)}{\sigma\gamma T} + \frac{\gamma LM}{2\sigma}. \tag{6.14}$$

Proof of Theorem 6.4.1: When γ_t is constant and $\gamma_t = \gamma$, taking expectation of (6.11) in Lemma 6.4.1, we obtain:

$$\mathbb{E}\left[f(w^{t+1})\right] - \mathbb{E}\left[f(w^t)\right] \leq -\sigma\gamma \mathbb{E}\left\| \nabla f(w^t) \right\|_2^2 + \frac{\gamma^2 LM}{2}, \tag{6.15}$$

Summing (6.15) from $t = 0$ to $T - 1$, we have:

$$\mathbb{E}\left[f(w^T)\right] - f(w^0) \leq -\sigma\gamma \sum_{t=0}^{T-1} \mathbb{E}\left\| \nabla f(w^t) \right\|_2^2 + \frac{\gamma^2 LMT}{2}. \tag{6.16}$$

Suppose w^* is the optimal solution for $f(w)$, therefore $f(w^*) - f(w^0) \leq \mathbb{E}\left[f(w^T)\right] - f(w^0)$. Above all, the following inequality is guaranteed that:

$$\frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}\left\| \nabla f(w^t) \right\|_2^2 \leq \frac{f(w^0) - f(w^*)}{\sigma\gamma T} + \frac{\gamma LM}{2\sigma}. \tag{6.17}$$

□

Therefore, the best solution we can obtain is controlled by $\frac{\gamma LM}{2\sigma}$. We also prove that Algorithm 11 can guarantee the convergence to critical points for the non-convex problem, as long as the diminishing learning rates satisfy the requirements in [84] such that:

$$\lim_{T\to\infty}\sum_{t=0}^{T-1}\gamma_t = \infty \quad \text{and} \quad \lim_{T\to\infty}\sum_{t=0}^{T-1}\gamma_t^2 < \infty. \tag{6.18}$$

Theorem 6.4.2. *Assume that Assumptions 6.4.1 and 6.4.2 hold and the diminishing learning rate sequence $\{\gamma_t\}$ satisfies (6.18). In addition, we assume w^* to be the optimal solution to $f(w)$. Setting $\Gamma_T = \sum_{t=0}^{T-1}\gamma_t$, then the output of Algorithm 11 satisfies that:*

$$\frac{1}{\Gamma_T}\sum_{t=0}^{T-1}\gamma_t\mathbb{E}\left\|\nabla f(w^t)\right\|_2^2 \;\leq\; \frac{f(w^0)-f(w^*)}{\sigma\Gamma_T} + \frac{LM}{2\sigma}\frac{\sum_{t=0}^{T-1}\gamma_t^2}{\Gamma_T}. \tag{6.19}$$

Proof of Theorem 6.4.2: Taking total expectation of (6.11) in Lemma 5.4.1 and summing it from $t=0$ to $T-1$, we obtain:

$$\mathbb{E}\left[f(w^T)\right] - f(w^0) \;\leq\; -\sigma\sum_{t=0}^{T-1}\gamma_t\mathbb{E}\left\|\nabla f(w^t)\right\|_2^2 + \frac{LM}{2}\sum_{t=0}^{T-1}\gamma_t^2. \tag{6.20}$$

Suppose w^* is the optimal solution for $f(w)$, therefore $f(w^*) - f(w^0) \leq \mathbb{E}\left[f(w^T)\right] - f(w^0)$. Letting $\Gamma_T = \sum_{t=0}^{T-1}\gamma_t$, we have:

$$\frac{1}{\Gamma_T}\sum_{t=0}^{T-1}\gamma_t\mathbb{E}\left\|\nabla f(w^t)\right\|_2^2 \;\leq\; \frac{f(w^0)-f(w^*)}{\sigma\Gamma_T} + \frac{LM}{2\sigma}\frac{\sum_{t=0}^{T-1}\gamma_t^2}{\Gamma_T}. \tag{6.21}$$

We complete the proof.

□

Remark 6.4.1. *Suppose w^s is chosen randomly from $\{w^t\}_{t=0}^{T-1}$ with probabilities proportional to $\{\gamma_t\}_{t=0}^{T-1}$. According to Theorem 6.4.2, we can prove that Algorithm 11 guarantees convergence to critical points for the non-convex problem:*

$$\lim_{s\to\infty}\mathbb{E}\|\nabla f(w^s)\|_2^2 \;=\; 0\,. \tag{6.22}$$

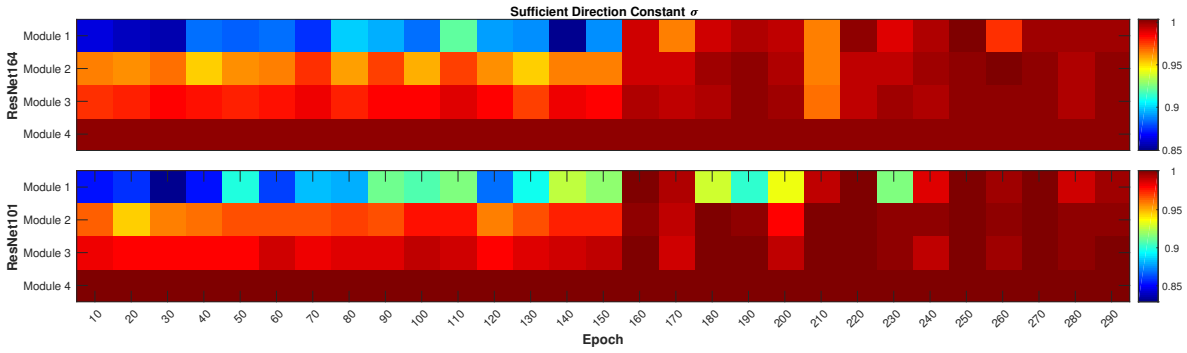


Figure 26: Sufficient direction constant for ResNet164 and ResNet101 on CIFAR-10.

6.5 Experimental Results

We validate our method with experiments training deep convolutional neural networks. Experimental results show that the proposed method achieves **faster** convergence, **lower** memory consumption and **better** generalization error than compared methods.

6.5.1 Experimental Setting

Implementations: We implement our method in PyTorch [77], and evaluate it with ResNet models [30] on two image classification benchmark datasets: CIFAR-10 and CIFAR-100 [53]. We adopt the standard data augmentation techniques in [30, 37, 63] for training these two datasets: random cropping, random horizontal flipping and normalizing. We use SGD with the momentum of 0.9, and the learning rate is initialized to 0.01. Each model is trained using batch size 128 for 300 epochs and the learning rate is divided by a factor of 10 at 150 and 225 epochs. The weight decay constant is set to 5×10^{-4}. In the experiment, a neural network with K modules is sequentially distributed across K GPUs. All experiments are performed on a server with four Titan X GPUs.

Compared Methods: We compare the performance of four methods in the experiments, including:

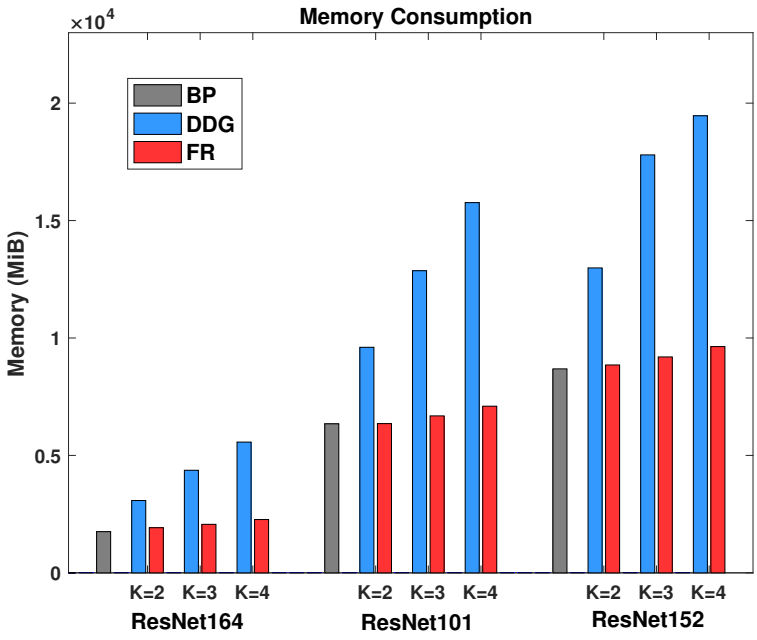


Figure 27: Memory consumption for ResNet164, ResNet101 and ResNet152.

• BP: we use the backpropagation algorithm [85] in PyTorch Library.

• DNI: we implement the decoupled neural interface in [42]. Following [42], the synthetic network has two hidden convolutional layers with 5×5 filters, padding of size 2, batch-normalization [41] and ReLU [72]. The output layer is a convolutional layer with 5×5 filters and padding size of 2.

• DDG: we implement the decoupled parallel backpropagation in [38].

• FR: features replay algorithm in Algorithm 11.

6.5.2 Sufficient Direction

We demonstrate that the proposed method satisfies Assumption 6.4.1 empirically. In the experiment, we divide ResNet164 and ResNet 101 into 4 modules and visualize the variations of the sufficient direction constant σ during the training period in Figure 26. Firstly, it is obvious that the values of σ of these modules are larger than 0 all the time. Therefore,

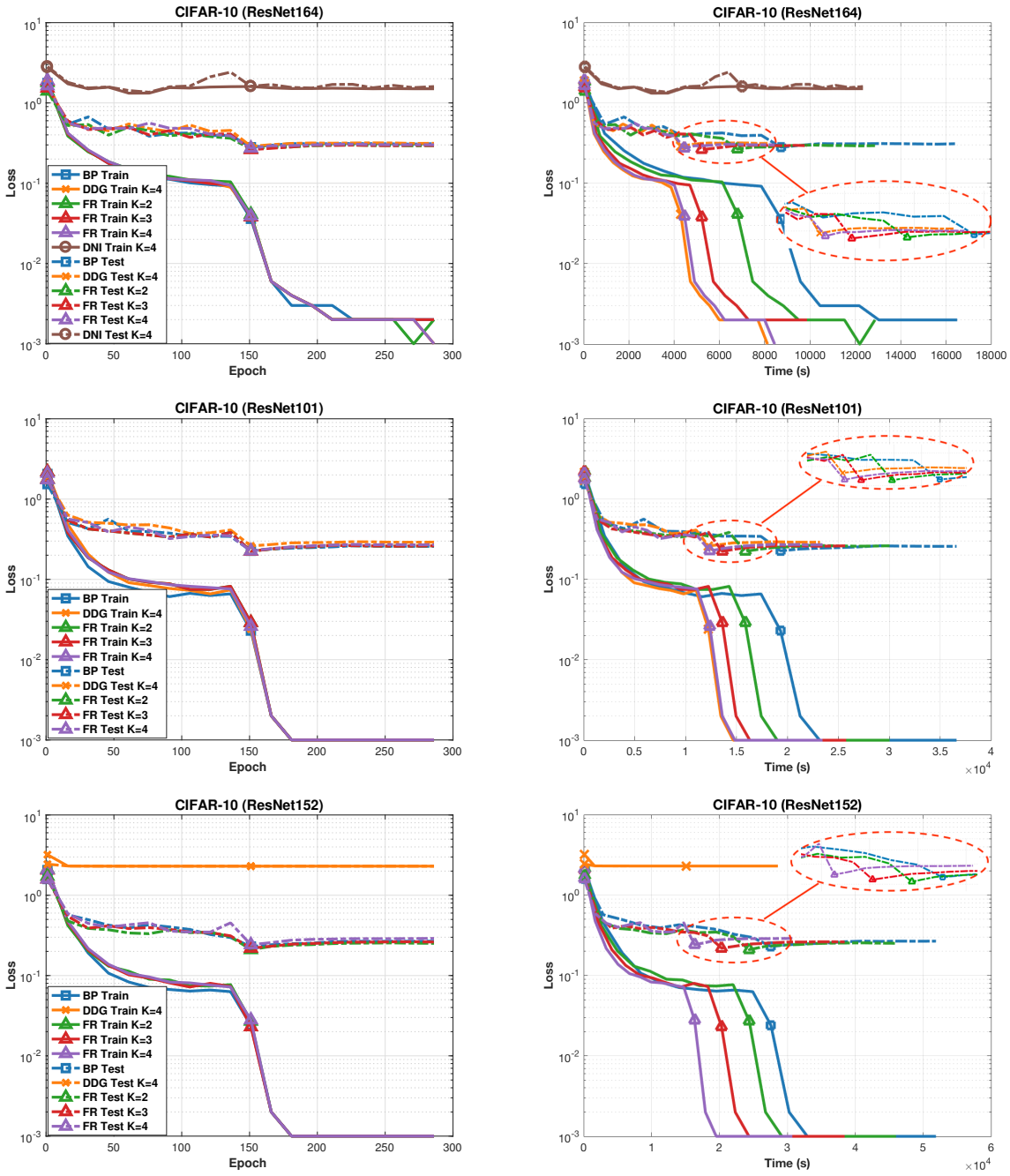


Figure 28: Training and testing curves for ResNet-164, ResNet101 and ResNet152 on CIFAR-10.

Table 6: Comparisons of memory consumption of the neural network with L layers, which is divided into K modules.

Algorithm	Backward Locking	Memory (Activations)
BP [85]	yes	$\mathcal{O}(L)$
DNI [42]	no	$\mathcal{O}(L + KL_s)$
DDG [38]	no	$\mathcal{O}(LK + K^2)$
FR	no	$\mathcal{O}(L + K^2)$

Assumption 6.4.1 is satisfied such that Algorithm 11 is guaranteed to converge to the critical points for the non-convex problem. Secondly, we can observe that the values of σ of the lower modules are relatively small at the first half epochs, and become close to 1 afterwards. The variation of σ indicates the difference between the descent direction of FR and the steepest descent direction. Small σ at early epochs can help the method escape from saddle points and find better local minimum; large σ at the final epochs can prevent the method from diverging. In the following context, we will show that our method has better generation error than compared methods.

6.5.3 Performance Comparisons

To evaluate the performance of the compared methods, we utilize three criterion in the experiment including convergence speed, memory consumption, and generalization error.

Faster Convergence: In the experiments, we evaluate the compared methods with three ResNet models: ResNet164 with the basic building block, ResNet101 and ResNet152 with the bottleneck building block [30]. The performances of the compared methods on CIFAR-10 are shown in Figure 28. There are several nontrivial observations as follows: Firstly, DNI cannot converge for all models. The synthesizer network in [42] is so small that it cannot learn an accurate approximation of the error gradient when the network is deep.

Table 7: Best testing error rates of the compared methods on CIFAR-10 and CIFAR-100 datasets.

Model	CIFAR [53]	BP [85]	DDG [38]	FR
ResNet164	C-10	6.40	6.45	**6.03**
	C-100	28.53	28.51	**27.34**
ResNet101	C-10	5.25	5.35	**4.97**
	C-100	23.48	24.25	**23.10**
ResNet152	C-10	5.26	5.72	**4.91**
	C-100	25.20	26.39	**23.61**

Secondly, DDG cannot converge for the model ResNet152 when we set $K = 4$. The stale gradients can impose noise in the optimization and lead to divergence. Thirdly, our method converges much faster than BP when we increase the number of modules. In the experiment, the proposed method FR can achieve a speedup of up to 2 times compared to BP. We do not consider data parallelism for BP in this section. In the supplementary material, we show that our method also converges faster than BP with data parallelism.

Lower Memory Consumption: In Figure 27, we present the memory consumption of the compared methods for three models when we vary the number of modules K. We do not consider DNI because it does not converge for all models. It is evident that the memory consumptions of FR and BP are very close. On the contrary, when $K = 4$, the memory consumption of DDG is more than two times of the memory consumption of BP. The observations in the experiment are also consistent with the analysis in Table 6. For DNI, since a three-layer synthesizer network cannot converge, it is reasonable to assume that L_s should be large if the network is very deep. We do not explore it because it is out of the scope of this chapter. We always set K very small such that $K \ll L$ and $K \ll L_s$. FR algorithm can still obtain a good speedup when K is very small according to the second row in Figure 28.

Better Generalization Error: Table 7 shows the best testing error rates for the compared methods. We do not report the result of DNI because it does not converge. We can observe that FR always obtains better testing error rates than other two methods BP and DDG by a large margin. We think it is related to the variation of the sufficient descent constant σ. Small σ at the early epochs help FR escape saddle points and find better local minimum, large σ at the final epochs prevent FR from diverging. DDG usually performs worse than BP because the stale gradients impose noise in the optimization, which is commonly observed in asynchronous algorithms with stale gradients [13].

7.0 Conclusion

In this thesis, we propose several novel distributed algorithms to address the challenges in big models and big data. Firstly, we propose and analyze asynchronous mini-batch gradient descent method with variance reduction for non-convex optimization on two distributed architectures: shared-memory architecture and distributed-memory architecture. We analyze their convergence rate and prove that both of them can get a convergence rate of $O(1/T)$ for non-convex optimization. Linear speedup is accessible when we increase the number of workers K, if K is upper bounded. Experiment results on real dataset also demonstrate our analysis.

Then, we proposed Distributed Asynchronous Dual Free Coordinate Ascent (Dis- dfSDCA) method for distributed machine learning. We addressed two challenging issues in previous primal-dual distributed optimization methods: firstly, Dis-dfSDCA does not rely on the dual formulation, and can be used to solve the non-convex problem; secondly, Dis-dfSDCA uses asynchronous communication and can be applied on the complicated distributed system where there is straggler problem. We also analyze the convergence rate of Dis-dfSDCA and prove linear convergence even if the loss functions are non-convex, as long as the sum of non-convex objectives is convex. We conduct experiments on the simulated distributed system with straggler problem, and results consistently verify our theoretical analysis.

We also propose a novel Complete Layer-wise Adaptive Rate Scaling (CLARS) algorithm to remove warmup in the large-batch deep learning training. Then, we introduce fine-grained analysis and prove the convergence of the proposed algorithm for non-convex problems. Based on our analysis, we bridge the gap between several large-batch deep learning optimization heuristics and theoretical underpins. Extensive experiments demonstrate that the proposed algorithm outperforms gradual warmup by a large margin and defeats the convergence of the state-of-the-art large-batch optimizer (LARS) in training advanced deep neural networks on ImageNet dataset.

If the model is too big to train on a single device, we propose decoupled parallel backpropagation algorithm, which breaks the backward locking in backpropagation algorithm using

delayed gradients. We then apply the decoupled parallel backpropagation to two stochastic methods for deep learning optimization. In the theoretical section, we also provide convergence analysis and prove that the proposed method guarantees convergence to critical points for the non-convex problem. Finally, we perform experiments on deep convolutional neural networks, results verifying that our method can accelerate the training significantly without loss of accuracy.

To reduce the memory consumption and improve generalization error, we proposed a novel parallel-objective formulation for the objective function of the neural network and broke the backward locking using a new features replay algorithm. Besides the new algorithms, our theoretical contributions include analyzing the convergence property of the proposed method and proving that our new algorithm is guaranteed to converge to critical points for the non-convex problem under certain conditions. We conducted experiments with deep convolutional neural networks on two image classification datasets, and all experimental results verify that the proposed method can achieve faster convergence, lower memory consumption, and better generalization error than compared methods.

Bibliography

[1] Martın Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[2] Alekh Agarwal and John C Duchi. Distributed delayed stochastic optimization. In *Advances in Neural Information Processing Systems*, pages 873–881, 2011.

[3] Takuya Akiba, Shuji Suzuki, and Keisuke Fukuda. Extremely large minibatch sgd: training resnet-50 on imagenet in 15 minutes. *arXiv preprint arXiv:1711.04325*, 2017.

[4] Zeyuan Allen-Zhu and Elad Hazan. Variance reduction for faster non-convex optimization. In *International conference on machine learning*, pages 699–707, 2016.

[5] Zeyuan Allen-Zhu and Yang Yuan. Improved svrg for non-strongly-convex or sum-of-non-convex objectives. In *International conference on machine learning*, pages 1080–1089, 2016.

[6] David Balduzzi, Hastagiri Vanchinathan, and Joachim M Buhmann. Kickback cuts backprop's red-tape: Biologically plausible credit assignment in neural networks. In *AAAI*, pages 485–491, 2015.

[7] Amir Beck and Luba Tetruashvili. On the convergence of block coordinate descent type methods. *SIAM journal on Optimization*, 23(4):2037–2060, 2013.

[8] Yoshua Bengio et al. Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127, 2009.

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

[10] Léon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *Siam Review*, 60(2):223–311, 2018.

[11] Miguel Carreira-Perpinan and Weiran Wang. Distributed optimization of deeply nested systems. In *Artificial Intelligence and Statistics*, pages 10–19, 2014.

[12] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[13] Jianmin Chen, Xinghao Pan, Rajat Monga, Samy Bengio, and Rafal Jozefowicz. Revisiting distributed synchronous sgd. *arXiv preprint arXiv:1604.00981*, 2016.

[14] Dominik Csiba, Zheng Qu, and Peter Richtárik. Stochastic dual coordinate ascent with adaptive probabilities. In *International Conference on Machine Learning*, pages 674–683, 2015.

[15] Wojciech Marian Czarnecki, Grzegorz Świrszcz, Max Jaderberg, Simon Osindero, Oriol Vinyals, and Koray Kavukcuoglu. Understanding synthetic gradients and decoupled neural interfaces. *arXiv preprint arXiv:1703.00522*, 2017.

[16] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, et al. Large scale distributed deep networks. In *Advances in Neural Information Processing Systems*, pages 1223–1231, 2012.

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

[18] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. Ieee, 2009.

[19] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, 2019.

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

[21] Ronen Eldan and Ohad Shamir. The power of depth for feedforward neural networks. In *Conference on Learning Theory*, pages 907–940, 2016.

[22] Edgar Gabriel, Graham E. Fagg, George Bosilca, Thara Angskun, Jack J. Dongarra, Jeffrey M. Squyres, Vishal Sahay, Prabhanjan Kambadur, Brian Barrett, Andrew Lumsdaine, Ralph H. Castain, David J. Daniel, Richard L. Graham, and Timothy S. Woodall. Open MPI: Goals, concept, and design of a next generation MPI implementation. In *Proceedings, 11th European PVM/MPI Users' Group Meeting*, pages 97–104, Budapest, Hungary, September 2004.

[23] Dan Garber and Elad Hazan. Fast and simple pca via convex optimization. *arXiv preprint arXiv:1509.05647*, 2015.

[24] Saeed Ghadimi and Guanghui Lan. Stochastic first-and zeroth-order methods for nonconvex stochastic programming. *SIAM Journal on Optimization*, 23(4):2341–2368, 2013.

[25] Saeed Ghadimi and Guanghui Lan. Accelerated gradient methods for nonconvex nonlinear and stochastic programming. *Mathematical Programming*, 156(1-2):59–99, 2016.

[26] Saeed Ghadimi, Guanghui Lan, and Hongchao Zhang. Mini-batch stochastic approximation methods for nonconvex stochastic composite optimization. *Mathematical Programming*, 155(1-2):267–305, 2016.

[27] Akhilesh Gotmare, Nitish Shirish Keskar, Caiming Xiong, and Richard Socher. A closer look at deep learning heuristics: Learning rate restarts, warmup and distillation. *International Conference on Learning Representations*, 2018.

[28] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

[29] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[30] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

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

[32] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Lecture 6a overview of mini–batch gradient descent. *Coursera Lecture slides https://class. coursera. org/neuralnets-2012-001/lecture,[Online*, 2012.

[33] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. *Cited on*, page 14, 2012.

[34] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[35] Elad Hoffer, Itay Hubara, and Daniel Soudry. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. In *Advances in Neural Information Processing Systems*, pages 1731–1741, 2017.

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

[37] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[38] Zhouyuan Huo, Bin Gu, Qian Yang, and Heng Huang. Decoupled parallel backpropagation with convergence guarantee. *arXiv preprint arXiv:1804.10574*, 2018.

[39] Zhouyuan Huo and Heng Huang. Asynchronous mini-batch gradient descent with variance reduction for non-convex optimization. In *AAAI*, pages 2043–2049, 2017.

[40] Zhouyuan Huo, Xue Jiang, and Heng Huang. Asynchronous dual free stochastic dual coordinate ascent for distributed data mining. In *2018 IEEE International Conference on Data Mining (ICDM)*, pages 167–176. IEEE, 2018.

[41] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.

[42] Max Jaderberg, Wojciech Marian Czarnecki, Simon Osindero, Oriol Vinyals, Alex Graves, David Silver, and Koray Kavukcuoglu. Decoupled neural interfaces using synthetic gradients. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1627–1635. JMLR. org, 2017.

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

[44] Xianyan Jia, Shutao Song, Wei He, Yangzihao Wang, Haidong Rong, Feihu Zhou, Liqiang Xie, Zhenyu Guo, Yuanzhou Yang, Liwei Yu, et al. Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes. *arXiv preprint arXiv:1807.11205*, 2018.

[45] Justin Johnson. Benchmarks for popular cnn models. `https://github.com/jcjohnson/cnn-benchmarks`, 2017.

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

[47] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 655–665, 2014.

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

[49] Yoon Kim. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751, 2014.

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

[51] Jakub Konečnỳ, Jie Liu, Peter Richtárik, and Martin Takáč. Mini-batch semi-stochastic gradient descent in the proximal setting. *IEEE Journal of Selected Topics in Signal Processing*, 10(2):242–255, 2015.

[52] Alex Krizhevsky. One weird trick for parallelizing convolutional neural networks. *arXiv preprint arXiv:1404.5997*, 2014.

[53] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images, 2009.

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

[55] Guanghui Lan. An optimal method for stochastic composite optimization. *Mathematical Programming*, 133(1-2):365–397, 2012.

[56] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[57] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[58] Mu Li. *Scaling Distributed Machine Learning with System and Algorithm Co-design*. PhD thesis, PhD thesis, Intel, 2017.

[59] Xiangru Lian, Yijun Huang, Yuncheng Li, and Ji Liu. Asynchronous parallel stochastic gradient for nonconvex optimization. In *Advances in Neural Information Processing Systems*, pages 2719–2727, 2015.

[60] Xiangru Lian, Huan Zhang, Cho-Jui Hsieh, Yijun Huang, and Ji Liu. A comprehensive linear speedup analysis for asynchronous stochastic parallel optimization from zeroth-order to first-order. In *Advances in Neural Information Processing Systems*, pages 3054–3062, 2016.

[61] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[62] Hongzhou Lin, Julien Mairal, and Zaid Harchaoui. A universal catalyst for first-order optimization. In *Advances in Neural Information Processing Systems*, pages 3384–3392, 2015.

[63] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

[64] Ji Liu, Stephen J Wright, and Srikrishna Sridhar. An asynchronous parallel randomized kaczmarz algorithm. *arXiv preprint arXiv:1401.4780*, 2014.

[65] Chenxin Ma, Jakub Konečnỳ, Martin Jaggi, Virginia Smith, Michael I Jordan, Peter Richtárik, and Martin Takáč. Distributed optimization with arbitrary local solvers. *Optimization Methods and Software*, pages 1–36, 2017.

[66] 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-Volume 37*, pages 1973–1982, 2015.

[67] Dhruv Mahajan, Ross Girshick, Vignesh Ramanathan, Kaiming He, Manohar Paluri, Yixuan Li, Ashwin Bharambe, and Laurens van der Maaten. Exploring the limits of weakly supervised pretraining. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 181–196, 2018.

[68] Hiroaki Mikami, Hisahiro Suganuma, Yoshiki Tanaka, Yuichi Kageyama, et al. Imagenet/resnet-50 training in 224 seconds. *arXiv preprint arXiv:1811.05233*, 2018.

[69] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937, 2016.

[70] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[71] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg

Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[72] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[73] Arkadi Nemirovski, Anatoli Juditsky, Guanghui Lan, and Alexander Shapiro. Robust stochastic approximation approach to stochastic programming. *SIAM Journal on optimization*, 19(4):1574–1609, 2009.

[74] Yurii E Nesterov. A method for solving the convex programming problem with convergence rate o (1/kˆ 2). In *Dokl. akad. nauk Sssr*, volume 269, pages 543–547, 1983.

[75] Jiquan Ngiam, Adam Coates, Ahbik Lahiri, Bobby Prochnow, Quoc V Le, and Andrew Y Ng. On optimization methods for deep learning. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 265–272, 2011.

[76] Arild Nøkland. Direct feedback alignment provides learning in deep neural networks. In *Advances in Neural Information Processing Systems*, pages 1037–1045, 2016.

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

[78] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural networks*, 12(1):145–151, 1999.

[79] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 1:8, 2019.

[80] 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*, pages 693–701, 2011.

[81] Sashank J Reddi, Ahmed Hefny, Suvrit Sra, Barnabás Póczos, and Alex Smola. Stochastic variance reduction for nonconvex optimization. In *International conference on machine learning*, pages 314–323, 2016.

[82] Sashank J Reddi, Ahmed Hefny, Suvrit Sra, Barnabás Póczos, and Alex J Smola. On variance reduction in stochastic gradient descent and its asynchronous variants. In *Advances in Neural Information Processing Systems*, pages 2629–2637, 2015.

[83] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[84] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

[85] David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

[86] Conrad Sanderson and Ryan Curtin. Armadillo: a template-based c++ library for linear algebra. *Journal of Open Source Software*, 2016.

[87] Cicero D Santos and Bianca Zadrozny. Learning character-level representations for part-of-speech tagging. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1818–1826, 2014.

[88] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization? In *Advances in Neural Information Processing Systems*, pages 2483–2493, 2018.

[89] Shai Shalev-Shwartz. Sdca without duality, regularization, and individual convexity. In *International Conference on Machine Learning*, pages 747–754, 2016.

[90] Shai Shalev-Shwartz. Sdca without duality, regularization, and individual convexity. In *International Conference on Machine Learning*, pages 747–754, 2016.

[91] Shai Shalev-Shwartz and Tong Zhang. Accelerated mini-batch stochastic dual coordinate ascent. In *Advances in Neural Information Processing Systems*, pages 378–385, 2013.

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

[93] Christopher J Shallue, Jaehoon Lee, Joseph Antognini, Jascha Sohl-Dickstein, Roy Frostig, and George E Dahl. Measuring the effects of data parallelism on neural network training. *Journal of Machine Learning Research*, 20:1–49, 2019.

[94] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

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

[96] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

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

[98] Gavin Taylor, Ryan Burmeister, Zheng Xu, Bharat Singh, Ankit Patel, and Tom Goldstein. Training neural networks without gradients: A scalable admm approach. In *International Conference on Machine Learning*, pages 2722–2731, 2016.

[99] Matus Jan Telgarsky. Benefits of depth in neural networks. *Journal of Machine Learning Research*, 49(June):1517–1539, 2016.

[100] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[101] Omry Yadan, Keith Adams, Yaniv Taigman, and Marc'Aurelio Ranzato. Multi-gpu training of convnets. *arXiv preprint arXiv:1312.5853*, 2013.

[102] Tianbao Yang. Trading computation for communication: Distributed stochastic dual coordinate ascent. In *Advances in Neural Information Processing Systems*, pages 629–637, 2013.

[103] Tianbao Yang, Qihang Lin, and Zhe Li. Unified convergence analysis of stochastic momentum methods for convex and non-convex optimization. *arXiv preprint arXiv:1604.03257*, 2016.

[104] Dong Yin, Ashwin Pananjady, Max Lam, Dimitris Papailiopoulos, Kannan Ramchandran, and Peter Bartlett. Gradient diversity: a key ingredient for scalable distributed learning. In *International Conference on Artificial Intelligence and Statistics*, pages 1998–2007, 2018.

[105] Chris Ying, Sameer Kumar, Dehao Chen, Tao Wang, and Youlong Cheng. Image classification at supercomputer scale. *arXiv preprint arXiv:1811.06992*, 2018.

[106] Yang You, Igor Gitman, and Boris Ginsburg. Scaling sgd batch size to 32k for imagenet training. *arXiv preprint arXiv:1708.03888*, 6, 2017.

[107] Yang You, Jonathan Hseu, Chris Ying, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. Large-batch training for lstm and beyond. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–16, 2019.

[108] Ruiliang Zhang and James Kwok. Asynchronous distributed admm for consensus optimization. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1701–1709, 2014.

[109] Ruiliang Zhang, Shuai Zheng, and James T Kwok. Asynchronous distributed semi-stochastic gradient optimization. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[110] 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.

[111] Shen-Yi Zhao and Wu-Jun Li. Fast asynchronous parallel stochastic gradient descent: A lock-free approach with convergence guarantee. In *Thirtieth AAAI conference on artificial intelligence*, 2016.

[112] Martin Zinkevich, John Langford, and Alex J Smola. Slow learners are fast. In *Advances in neural information processing systems*, pages 2331–2339, 2009.

[113] Dongmian Zou, Radu Balan, and Maneesh Singh. On lipschitz bounds of general convolutional neural networks. *IEEE Transactions on Information Theory*, 2019.