Speeding up deep neural network training with decoupled and analytic learning

Zhuang, Huiping

2021

Zhuang, H. (2021). Speeding up deep neural network training with decoupled and analytic learning. Doctoral thesis, Nanyang Technological University, Singapore.
<https://hdl.handle.net/10356/153079>

<https://hdl.handle.net/10356/153079>

<https://doi.org/10.32657/10356/153079>

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License (CC BY-NC 4.0).

Downloaded on 26 Apr 2023 21:20:39 SGT

**Speeding Up Deep Neural Network Training with
Decoupled and Analytic Learning**

ZHUANG HUIPING

School of Electrical & Electronic Engineering

A thesis submitted to the Nanyang Technological University
in partial fulfilment of the requirement for the degree of
Doctor of Philosophy

2021

Statement of Originality

I hereby certify that the work embodied in this thesis is the result of original research, is free of plagiarised materials, and has not been submitted for a higher degree to any other University or Institution.

2021-Nov-03

.....

Date

NTU NTU NTU NTU NTU NTU NTU NTU
NTU NTU NTU NTU NTU NTU NTU NTU
NTU NTU NTU NTU NTU NTU NTU NTU
NTU NTU NTU NTU NTU NTU NTU NTU

.....
Zhuang Huiping

Supervisor Declaration Statement

I have reviewed the content and presentation style of this thesis and declare it is free of plagiarism and of sufficient grammatical clarity to be examined. To the best of my knowledge, the research and writing are those of the candidate except as acknowledged in the Author Attribution Statement. I confirm that the investigations were conducted in accord with the ethics policies and integrity standards of Nanyang Technological University and that the research data are presented honestly and without prejudice.

03 Nov. 2021

.....
Date

NTU NTU NTU NTU NTU NTU NTU NTU
NTU NTU NTU NTU NTU NTU NTU NTU
NTU NTU NTU NTU NTU NTU NTU NTU
NTU NTU NTU NTU NTU NTU NTU NTU
.....
Lin Zhiping

Authorship Attribution Statement

This thesis contains material from 3 papers published in the following peer-reviewed journals, and from 1 paper accepted at conference in which I am listed as an author.

Chapter 3 contains materials published in H. Zhuang, Y. Wang, Q. Liu and Z. Lin, “Fully Decoupled Neural Network Learning Using Delayed Gradients,” IEEE Trans. Neural Netw. Learn. Syst., doi: 10.1109/TNNLS.2021.3069883.

The contributions of the co-authors are as follows:

- I came up with the idea, did the derivations and the experiments.
- I wrote the drafts of the manuscript. The manuscript was revised by Prof. Lin Zhiping, Dr. Wang Yi and Dr. Liu Qinglai.

Chapter 4 contains materials published in H. Zhuang, Z. Weng, F. Luo, Z. Lin, K-A Toh and H. Li, “Mitigating Gradient Staleness in Decoupled Learning for Classification Tasks,” in International Conference of Machine Learning 2021.

The contributions of the co-authors are as follows:

- I came up with the idea, did the derivations and the experiments.
- I wrote the drafts of the manuscript. The manuscript was revised by Prof. Lin Zhiping, Prof Toh Kar-Ann, Prof Li Haizhou, Dr. Weng Zhenyu and Dr. Luo Fulin.

Chapter 5 contains materials published in H. Zhuang, Z. Lin and K-A Toh, “Correlation Projection for Analytic Learning of a Classification Network,” Neural Process. Lett.

The contributions of the co-authors are as follows:

- I came up with the idea, did the derivations and the experiments.
- I wrote the drafts of the manuscript. The manuscript was revised by Prof. Lin Zhiping and Prof. Toh Kar-Ann.

Chapter 6 contains materials published in H. Zhuang, Z. Lin and K-A Toh, "Blockwise Recursive Moore-Penrose Inverse for Network Learning," IEEE Trans. Syst. Man Cybern. Syst., doi: 10.1109/TSMC.2021.3064241.

The contributions of the co-authors are as follows:

- I came up with the idea, did the derivations and the experiments.
- I wrote the drafts of the manuscript. The manuscript was revised by Prof. Lin Zhiping and Prof. Toh Kar-Ann.

2021-Nov-03

.....
Date

ITU NTU NTU NTU NTU NTU NTU NTU
NTU NTU NTU NTU NTU NTU NTU NTU
ITU NTU NTU NTU NTU NTU NTU NTU
ITU NTU NTU NTU NTU NTU NTU NTU

.....
Zhuang Huiping

Acknowledgements

I would like to express my gratitude to all the people who have supported me through my Ph.D. journey. Thanks to their helps, my Ph.D. study has become more than just an academic pursuit.

First, I would like to sincerely thank my supervisor Prof. Lin Zhiping, who has been the most important guide academically and mentally through my Ph.D. career. He is kind and patient, and has provided me with more than enough resources for my research. He pays great attention to details. This has influenced me in a good way, and I have become more attentive, especially to the academic writing. I feel very lucky to be his student.

I would also like to express my deepest gratitude to Prof. Toh Kar-Ann from Yonsei University. He has supported me greatly in my research of analytic learning in which we have co-authored several good papers. He has spent a great amount of time helping me improve my writing skills. Thanks to Prof. Toh and Prof. Lin, I am lucky to receive a rather standard training of academic writing as a Ph.D. student.

Many thanks to my fellow Ph.D. friends, Cheng Deruo, Liu Qinglai, Luo Fulin, Ma Xiaoxi, Nan Shengyu, Oh Beomseok, Peng Xinggan, Peng Siyuan, Wang Yi, Weng Zhenyu, Zhou Lin and others. Special thanks to Dr. Liu Qianglai. He has helped me with many things and is one introducing me to Prof. Lin Zhiping so I have the

Acknowledgements

opportunity to receive my Ph.D. study in NTU.

I would also like to thank Thung Lina and Teng Siew Jiuan Catherine of the Communication Research Lab II, and Mr. Chia Chiu Kiat of the Information Systems Research Lab, for their excellent support during my Ph.D. study.

Finally yet most importantly, I would like to thank my family. My father Zhuang Shaorui and my mother Zhuang Peihua have sacrificed nearly everything to support the family's education, including mine thus far. They always believe me. Words cannot express my gratitude and love. I want to thank my brother Zhuang Huiya and my sisters Zhuang Hui, Zhuang Huimin and Zhuang Huici (yes very luckily my parents have five kids) for supporting our family and keeping our parents company in my absence. In particular, I would like to thank my beloved girlfriend (now my wife with a wedding away) Chen Xinyi who has been supportive from the beginning. She traveled across countries to stay with me. I would also like to thank my supervisor Porf. Lin Zhiping and Prof. Li Haizhou from National University of Singapore who helped make this happen.

Contents

Acknowledgements	i
Table of Contents	vii
Summary	viii
List of Figures	xiii
List of Tables	xvi
List of Abbreviations	xvii
1 Introduction	1
1.1 Motivation	4
1.1.1 Decoupled Learning	5
1.1.2 Analytic Learning	5
1.2 Objectives	7
1.3 Major Contribution of the Thesis	8
1.4 Organization of the Thesis	11
2 Literature Review	13

CONTENTS

2.1	Decoupled Learning	13
2.1.1	Local Error Learning Based Methods	14
2.1.2	Delayed Gradient Based Methods	16
2.1.3	Other Works Related to Decoupled Learning	17
2.2	Analytic Learning with Moore-Penrose Inverse	18
2.2.1	Analytic Learning Methods	18
2.2.2	MP Inverse Decomposition and Recursive MP Inverse Learning	20
3	Fully Decoupled Neural Network Learning Using Delayed Gradients	22
3.1	Preliminaries	23
3.2	Fully Decoupled Neural Network Learning	26
3.2.1	The Proposed FDG	26
3.2.2	The Gradient Shrinking Process	30
3.3	Convergence Analysis	33
3.4	Experiments	36
3.4.1	Comparison of Classification Performance	36
3.4.2	Comparison of Acceleration Performance	43
3.5	Summary	46
4	Accumulated Decoupled Learning with Gradient Staleness Mitiga-	
	tion	47
4.1	Preliminaries	49
4.1.1	Learning with Generic Gradient Accumulation	49
4.1.2	Gradient Staleness	50
4.2	An Accumulated Decoupled Learning	51
4.2.1	Asynchronous Learning without BP Lockings	52

CONTENTS

4.2.2	Update with Module-wise Gradient Accumulation	54
4.2.3	Impact of Module-wise Gradient Accumulation	54
4.2.4	Other Methods Using Delayed Gradients	57
4.3	Convergence Analysis	58
4.4	Experiments	60
4.4.1	Generalization Performance	61
4.4.2	Acceleration Performance	66
4.4.3	Memory Analysis	67
4.5	Summary	68
5	Correlation Projection for Analytic Learning	69
5.1	Preliminaries	70
5.1.1	Multilayer Network Structure	70
5.1.2	Multilayer Analytic Learning	72
5.2	Proposed Method	72
5.2.1	Training Phase	73
5.2.2	A Multi-exit Prediction Structure for Validation	80
5.2.3	The Testing Phase	83
5.2.4	Space Complexity Analysis	85
5.3	Experiments	86
5.3.1	UCI Datasets	88
5.3.2	ORL Face Recognition	90
5.3.3	MNIST Dataset	93
5.3.4	Runtime Comparison	94
5.4	Summary	97

CONTENTS

6	Block-wise Recursive Moore-Penrose Inverse for Network Learning	98
6.1	Preliminaries	99
6.1.1	The Linear Regression Problem	100
6.1.2	Moore-Penrose Inverse	101
6.1.3	The Memory Growing Problem and Recursive Methods	102
6.2	Block-wise Recursive Moore-Penrose Inverse	102
6.2.1	Block-wise Recursion by Decomposing the MP inverse	104
6.2.2	The Switching Structure and Linkage to RLS	112
6.3	Applications to Network Learning	117
6.3.1	Block-wise Formulation of the RBF Network	118
6.3.2	Block-wise Formulation of MLP Learning	119
6.3.3	Extension to Networks with Sparse Weights	124
6.4	Experiments	127
6.4.1	Solving Regression Problems	128
6.4.2	Solving Classification Problems	132
6.5	Summary	136
7	Conclusion and Recommendations	138
7.1	Conclusion	138
7.2	Recommendations for Further Research	142
	Author's Publications	144
	Bibliography	146
	Appendix A Proofs for the Theorems and Lemmas	160
A.1	Appendix for Chapter 3	160

CONTENTS

A.1.1	Proof of Theorem 3.3.1	160
A.1.2	Proof of Theorem 3.3.2	165
A.1.3	Proof of Theorem 3.3.3	166
A.2	Appendix for Chapter 4	167
A.2.1	Proof of Theorem 4.3.1	167
A.2.2	Proof of Theorem 4.3.2	171
A.2.3	Proof of Theorem 4.3.3	171
A.3	Appendix for Chapter 6	173
A.3.1	Proof for Theorem 6.2.1	173
A.3.2	Proof for Lemma 6.2.3	174
A.3.3	Proof for Lemma 6.2.4	175
A.3.4	Proof for Theorem 6.2.2	175
A.3.5	Proof for Lemma 6.2.5	176
A.3.6	Proof for Lemma 6.2.6	177
A.3.7	Proof for Corollary 6.2.2	178

Summary

Training deep neural networks usually demands a significantly long period of time. In this thesis, we explore methods in two different areas, i.e., decoupled learning and analytic learning, in order to reduce the training time.

In decoupled learning, new methods are proposed to alleviate the sequential nature of the backpropagation (BP) which accounts for the most common means of training deep neural networks. The BP requires a sequential passing of activations and gradients, which has been recognized as lockings (i.e., the forward, backward, and update lockings). These lockings impose strong synchronism among modules (a consecutive stack of layers), rendering most modules idle during training. A fully decoupled learning method using delayed gradients (FDG) is first proposed which addresses all the three lockings. The FDG improves training efficiency as a significant acceleration is achieved. Furthermore, the decoupled learning inevitably introduces asynchronism that causes gradient staleness (also known as stale gradient effect), resulting in degraded generalization performance or even divergence. An accumulated decoupled learning (ADL) is hence developed to cope with the staleness issue. The proposed ADL is proved to be effective in reducing the gradient staleness both theoretically and empirically, demonstrating an improved generalization ability compared with that of the current works which ignore the staleness.

New methods are also developed in the area of analytic learning by discarding the BP entirely and training the network using analytical solutions. The analytic learning trains neural networks in an exceedingly fast fashion as the training is completed within one single epoch. There are two main challenges in this area. The first challenge lies in the difficulty of finding analytical solutions for multilayer networks. Existing methods have several limitations, such as structural constraints or requesting invertible activation functions. Here a correlation projection network (CPNet) is developed which removes the aforementioned limitations by treating the network as a combination of multiple 2-layer modules. The analytic learning of CPNet is made possible after the label information is projected into the hidden modules so that each 2-layer module can analytically solve the locally supervised learning using the least squares solutions. The other challenge is that, to implement the analytic learning, there is a possible issue of memory leak caused by matrix operations based on the entire dataset. Hence, a block-wise recursive Moore-Penrose inverse (BRMP) method is proposed which can reformulate the original analytic learning exactly into a block-wise alternative using a block-wise decomposition of Moore-Penrose inverse. The BRMP not only reduces the memory consumption while keeping its high training efficiency, but also takes care of the potential rank-deficient matrix inversion issue during the analytic learning.

List of Figures

2.1	Diagram for an overview of the literature review.	14
3.1	Illustration of the proposed FDG: a $K = 3$ example. (a) By delaying the activations and gradients, the FDG allows the modules (in different colors) to be trained asynchronously in parallel. The communications among modules happen right before the iteration ends. It is observed that there is a delay of $2(K - k)$ of gradients in module k with a split of K . For instance, for $K = 3$, module $k = 1$ (gray) generates an activation at iteration 1 but the gradient of this batch arrives at iteration 5. (b) In the FDG, the backward pass (step B) is executed in the previously saved computation graph while the forward pass (step A) happens in the current one. This is how the FDG overcomes the sequential nature of the standard BP.	27
3.2	An intuitive interpretation for the GS process. (a) The DGs could cause the learning to miss the local minimum to a large margin. (b) By reducing the gradient strength, the learning could be brought closer to the local minimum than that without the GS process.	31

LIST OF FIGURES

3.3	(a) Performance of ResNet-18 trained on CIFAR-10 for various K with a maximum splittable number $K = 10$. (b) The difference between the GS process and the learning rate scaling (case of ResNet-18, $K=3$). (c)-(e) Impact of β on the performance conducted on ResNet-20.	39
3.4	(a) The left column images are validation error curves of the compared methods for WRN-28-10 on CIFAR-10 (the top/median/bottom images are for $K = 2/3/4$). (b)The right column images are corresponding training error curves (the top/median/bottom images are for $K = 2/3/4$).	41
4.1	Training a 3-layer network by ADL with $K = 3$ and mGA steps of (a) $M = 2$ and (b) $M = 4$, suggesting that each module is updated every 2 and 4 iterations respectively. Note that there is a batch index difference of $2(K - k)$ between the forward and backward pass. A larger M reduces the LoS according to Eq. (4.5). For instance, at $t = 5$ module 1 (gray) has LoS of 2 with $M = 2$, which is reduced to 1 with $M = 4$	51
4.2	The averaged LoS w.r.t. mGA step of M	56
4.3	Learning curves of compared methods for training ResNet-34 with $K = 12$ (the 1 st column) and ResNet-56 with $K = 16$ (the 2 nd column), and the error rates w.r.t. PS (the 3 rd column) on ResNet-18. Note that the DSP is excluded as no source code is available.	64

LIST OF FIGURES

5.1	The proposed CPNet has a module-based structure with each module containing two layers. (a) Training phase: The label information is first encoded into hidden layers through the label encoders. Subsequently, the weights in each module are determined by the CP process in the first layer and the LS solution in the second layer. (b) Prediction in the validation phase: Adopting the label decoders, each module can give its own prediction, achieving multiple exits for the same input.	74
5.2	An illustration of (a) validation phase determining the network structure (highlighted in the red dash box), and (b) testing phase evaluating the performance of the selected network. In this illustration, three network structures are evaluated with “ $M=m$, $H=h$ ” for hidden neuron sizes $h=100, 300, 1000$ on modules $m=1, \dots, 5$ in order to select the best validated network structure using only the training set. The network structure with $m=3$ and $h=300$ has been selected in view of its highest validation accuracy of 90%.	85
5.3	Samples from the ORL face recognition dataset.	90
5.4	Samples from the MNIST dataset.	91
5.5	Training time (seconds) comparison among the compared methods.	96
6.1	The structure of the proposed BRMP, which switches among Scenarios I-III (corresponding to Theorems 6.2.2, Corollaries 6.2.1 and 6.2.2) according to whether $\mathbf{Q}_{t-1} = \mathbf{0}$ (the kernel of the accumulated data is a null space), or $\mathbf{C}_t = \mathbf{0}$ (the arriving data is fully dependent on the accumulated data). In particular, Scenario III links the BRMP to the well-known RLS.	104

LIST OF FIGURES

6.2 Visualization of the decision boundaries of the RBF networks trained via (a) the batch-wise LS and (b) the proposed BRMP. The blue and red dots are the testing samples.	132
--	-----

List of Tables

3.1	Comparison with state-of-the-art methods in terms of lockings and auxiliary networks.	23
3.2	Table of Notations.	23
3.3	Ideal speedup of different decoupling methods for a network evenly split into K modules. $\mathcal{T}_f$, $\mathcal{T}_b$ and $\mathcal{T}_{aux}$ denote the computation time executing the forward, the backward pass, and the auxiliary network, respectively.	30
3.4	The validation errors of the compared methods on CIFAR-10 for $K = 2$. Results with * are rerun using the same training strategy in this chapter. Note that ResNet-18 has more parameters than ResNet-20, ResNet-56 and ResNet-110 as it has more CNN channels [5].	37
3.5	The validation errors of the compared methods on CIFAR-100 for $K = 2$. Results with * are rerun using the same training strategy in this chapter.	38
3.6	The validation errors of the compared methods on ImageNet for $K = 2$. Results with * are rerun using the same training strategy in this chapter.	38
3.7	Top 1 errors of ResNet-56 and WRN-28-10 on CIFAR-10 for $K=2, 3, 4$.	39

LIST OF TABLES

3.8	Speed comparisons of ResNet-101 among the compared methods on ImageNet. The communication ratio (Cm. ratio) is also given to measure the communication cost in the proposed FDG.	44
4.1	Characteristics of decoupled leaning methods and the proposed ADL. .	48
4.2	Table of Notations.	48
4.3	Validation errors (%) for the compared methods training various networks on CIFAR-10. $\otimes$ indicates divergence.	61
4.4	Validation errors (%) for the compared methods training various networks on CIFAR-10. $\otimes$ indicates divergence. The best results are marked by boldface black digits while the second best marked by boldface blue numbers.	63
4.5	Top1 and Top5 validation errors (%) of BP and ADL on ImageNet. . .	65
4.6	Speedups (over BP) in training ResNet-101 (ImageNet).	66
4.7	Memory consumption of the compared methods.	67
5.1	Table of Notations.	70
5.2	UCI dataset specifications.	88
5.3	Average testing accuracies (%) of compared methods on UCI datasets. .	89
5.4	Performance on ORL face dataset (N.A. means <i>not applicable</i>).	91
5.5	Performance on MNIST dataset.	92
6.1	Table of Notations.	99
6.2	Overall comparison between the batch-wise method and the BRMP for $K = 700$, block size 700 and $N = 10000$ by 5 independent runs given randomly generated data.	128

LIST OF TABLES

6.3	Residual error and time consumption at various block sizes (from 1 to 3000).	129
6.4	Residual error under rank-deficient data matrix (rank from 20 to 600).	129
6.5	Residual error at various sparse rates (from 20% to 99.99%).	130
6.6	Residual error and memory consumption at various dataset sizes (from 20k to 400k, where "1k" denotes 1000).	130
6.7	Time and memory usage w.r.t. the feature-sample ratio.	131
6.8	Dataset specifications.	133
6.9	MLP learning based on BRMP-KAR in comparison with KARnet. The SGD-BP is also included for the comparison of learning efficiency. Note that the hyperparameters are not tuned to maximize the classification performances and the measurements of time and memory are subject to slight fluctuation.	134

List of Abbreviations

ADL	Accumulated decoupled learning
ASGD	Asynchronous stochastic gradient descent
BP	Backpropagation
BRMP	Block-wise recursive Moore-Penrose inverse
CNN	Convolutional neural network
CP	Correlation projection
CPNet	Correlation projection network
DDG	Decoupled parallel BP using delayed gradients
DGL	Decoupled greedy learning
DSP	Diversely stale parameters
DG	Delayed gradient
DNI	Decoupled neural interface
ELM	Extreme learning machine
FA	Feedback alignment
F-MNIST	Fashion-MNIST dataset
FDG	Fully decoupled training scheme using delayed gradients
FF	Feedforward design
FR	Feature replay
gGA	Generic gradient accumulation
GS	Gradient shrinking
KARnet	kernel-and-range network
LD	label decoding

LE	label encoding
LS	Least squares
LEL	Local error learning
LHS	Left hand size
LoS	Level of staleness
mGA	Module-wise gradient accumulation
MLP	Multilayer perceptron
MP	Moore-Penrose
OOM	Out-of-memory
PS	Partition size
PCA	Principal component analysis
PILAE	Pseudoinverse autoencoder
PIL	Pseudoinverse learning
RBF	Radial basis function
RBM	Restricted Boltzmann machines
RHS	Right hand size
RLS	Recursive least squares
RMP	Recursive Moore-Penrose
RNN	Recurrent neural network
SGD	Stochastic gradient descent
w.r.t.	With respect to
WRN	Wide ResNet

Chapter 1

Introduction

Deep neural networks (DNNs), such as multilayer perceptron (MLP) [1] and convolutional neural network (CNN) [2], have demonstrated great success in various tasks, such as classification [3] and object detection [4]. In general, the performance of DNNs grows with deeper [5] and wider [6] structures. This, however, has led to an enormous growth of training time. There have been various attempts to reduce the consumed time for training DNNs. These attempts may be broadly categorized into two groups, namely the *parallel learning* and the *reformulation-based learning*.

Methods of parallel learning seek acceleration through parallelizing the training of DNNs, which mainly include data parallelism [7] and model parallelism [8]. Data parallelism techniques assign copies of the neural network into various workers. These workers respectively take separate subsets from the training data for learning processes, and engage gradient synchronization on the learnable parameters among themselves. Data parallelism has been well-explored. It thrives in the training of networks that are computation-intensive without too many trainable parameters (e.g., ResNet [5]), but becomes less efficient for parameter-intensive network training [7]

(e.g., VGG-19 [9]). Methods belonging to model parallelism partition the network into several parts with each part handled by one specific worker. Unlike data parallelism, the model parallelism eliminates the inefficient parameter synchronization process, and it spawns various forms. For instance, model parallelization can be adopted in CNNs by parallelizing the convolution operations [10].

An important branch of model parallelism is *decoupled learning*, which is a relatively new branch of research with its first contribution made only a few years ago in 2017 [11]. Decoupled learning partitions a network in a simple way by splitting it depth-wise into multiple modules—with each module containing a consecutive stack of layers—to facilitate inter-layer module-wise parallelization. The aim of decoupled learning is to address the so-called lockings (i.e., forward, backward, and update lockings [11]) originated in the conventional backpropagation (BP) algorithm. The BP requires a complete forward pass followed by a backward pass before the parameters are updated, which easily leads to inefficiency [12] especially for training deeper networks. Existing lockings keep the majority of the modules on hold during training, leading to learning inefficiency. For instance, for training a 2-layer neural network, when the computation (e.g., calculating gradients) is happening in the second layer, the first layer is idle (e.g., waiting for the second layer to finish its computation).

The literature of decoupled learning can be roughly categorized into two classes: delayed gradient (DG) based methods and local error learning (LEL) based methods. The DG-based methods (e.g., [13,14]) adopt DGs (a form of older gradients) to avoid the synchronism of BP, thereby breaking the lockings. They deliver comparable generalization performance to the BP baseline, but introduce instability in the form of *gradient staleness* [15] (or stale gradient effect). The resulted gradient staleness is mainly caused by utilizing older and hence less accurate gradients, and could lead to

decrease in generalization performance or even divergence during network learning. The LEL-based methods, on the other hand, achieve decoupled learning via severing the gradient flow between adjacent modules by building auxiliary networks to generate local error gradients. This allows the modules to avoid the global BP thereby solving all the three BP lockings. Methods in this class are distinguished by the design of auxiliary networks. The LEL-based methods do not invite gradient staleness, but usually encounter higher degree of generalization decline. This is because earlier modules trained by the LEL-based methods are entirely governed by auxiliary networks, and separated from the global information at the final layer.

Another way to improve the training efficiency is through altering the learning process. For instance, neural network pruning [16] alters its parameter setting during network learning. It gradually prunes (or removes) network parameters that are of little significance to the final decision making. In such a way, the total number of learnable parameters is reduced, thereby leading to speedup of network learning.

In a broader sense, on the other hand, the BP itself has several fundamental limitations. Applications of the BP mechanism often encounter issues related to search convergence. Firstly, updating of network parameters through BP could be prone to vanishing or exploding gradients. Secondly, as the learning is iteration-based, an inappropriate initialization may lead to slow convergence. Thirdly, BP learning often requires hundreds of visits or beyond, of an entire dataset (i.e., many epochs), which is time-consuming and resource-demanding. Finally, tuning of hyperparameters, such as learning rate scheduling and optimizer selection, could be tiresome without proper guidelines. This has led to the birth of *analytic learning* [17–20]. Such a learning paradigm engages a revolutionized reformulation of learning by discarding the BP entirely, and replacing it with closed-form solutions mainly characterized by least

1.1. MOTIVATION

squares (LS) formulations. Apart from bypassing the aforementioned limitations of BP, analytic learning also enjoys higher interpretability owing to the straightforward matrix manipulations during training [17, 21].

Analytic learning started with shallow networks (also known as 2-layer networks) [22–28]. In general, they determine the second layer weights using a closed-form LS solution with certain initialization in the first layer. An example is the random-weighted network [29, 30], which assigns random values to the weights in the first layer and then map the output to the label using a simple LS estimator. The progress of analytic learning over recent years has shown an increasing attention towards incorporation of the fast-learning attribute to multilayer neural networks¹. The challenge of such development mainly lies in the learning of hidden layers. One possible solution is to feed the label information into hidden layers to facilitate locally supervised learning. For instance, a kernel-and-range network (KARnet) proposed in [18] can analytically solve multilayer network learning with a sequential label projection into the hidden layers.

1.1 Motivation

In this thesis, among the pool of methods for accelerating the training of DNNs, efforts are made for the exploration of decoupled learning and analytic learning. This is in view of their various existing gaps accompanying promising research potentials.

¹The term *multilayer neural network* used in many literature [18] is also referred to as the DNN. In this thesis, these names will be used interchangeably.

1.1.1 Decoupled Learning

Unlike other model parallelism counterparts that usually demand an extensive alteration for realization, a simple depth-wise partition of decoupled learning encourages a straightforward implementation on various platforms with minimum effort. As an immature branch of research, decoupled learning is still in demand of further exploration in order to achieve its full potential. Although several attempts have the learning decoupled, they exhibit unresolved limitations. For instance, the decoupled parallel BP using delayed gradients (DDG) [13] and the feature replay (FR) [14], which belong to the DG-based methods, attain decoupled learning by removing the backward locking. However, as they are still constrained by the forward and the update lockings, the DDG and FR achieved very limited acceleration over BP (e.g., $2\times$ with 4 GPUs) in training DNNs. These DG-based methods are also affected by the gradient staleness that could cause performance degradation and instability. Alternatively, there are methods (mainly from the LEL-based methods) solving all the three BP lockings (i.e., locking-free), which allow a much more promising speedup. However, they either suffer from significant loss of generalization abilities [11], or are restricted to relatively shallow structures [31]. That is, in the area of decoupled learning, there remains a clear gap of seeking locking-free methods which can handle various DNN structures (e.g., various depths) without sacrificing their generalization performance. The focus here is to develop new decoupled learning techniques with high efficiency, flexibility, accuracy and robustness.

1.1.2 Analytic Learning

Unlike the BP training paradigm, methods in analytic learning typically convert a nonlinear network learning problem into linear segments, which are then tackled by

1.1. MOTIVATION

solving linear matrix equations. Analytic learning has been shown to thrive in training shallow networks [23–28]. However, in terms of multilayer neural networks, the formulation towards these analytical solutions remains difficult and illusive. There have been indeed several contributions [17, 21, 32], yet with rather unwelcome limitations (e.g., structural constraints and requesting invertible activation function). For instance, the pseudoinverse learning (PIL) [17] provides an analytic solution for training MLPs, but needs the hidden layer size to be matched with the size of the given training set. The KARnet [18] attempts to analytically solve the multilayer network learning, but it is conditioned on an invertible activation function. In addition, a technique called feedforward design (FF) [21] achieves analytic learning utilizing principal component analysis (PCA) for preprocessing, which, however, results in extra complexity and memory consumption. In summary, existing attempts of analytic learning either require invertible activation functions or suffer from certain structural constraints for multilayer network learning. This has motivated us to develop a new multilayer analytic learning method that is free from these constraints.

On the other hand, analytic learning is enabled mostly due to solving the LS problem based on the Moore-Penrose (MP) inverse² [33]. An important issue for implementing the MP inverse learning lies in the memory demand. For the BP training, the mini-batch stochastic gradient descent (SGD) [34] is a common trick to avoid out-of-memory (OOM) caused by the batch-wise learning (e.g., gradient descent) that processes the entire dataset during each update. Analytic learning [21, 22, 35–38] is also subjected to the OOM problem due to performing the MP inverse based on the entire data matrix (this may be referred to as the *batch-wise* computation). One possible solution is through decomposing the MP inverse to develop a recursive replacement

²The MP inverse is a form of generalized matrix inverse, and is also called pseudoinverse in many references.

for the batch-wise LS. In [39, 40], the recursive MP inverse [41] is adopted and reformulated into a recursive learning scheme. However, the recursive MP inverse based methods are sample-wise where the training samples are processed one-by-one in each iteration. Employing these sample-wise approaches leads to significantly longer execution time due to their sequential processing requirement. This has motivated us to develop a new formulation beyond the sample-wise capacity to improve the efficiency while preserving the low-memory property.

1.2 Objectives

Motivated by the aforementioned research gaps, the objectives of this thesis are to develop new methods to accelerate the training of DNNs, specifically in the areas of decoupled learning and analytic learning. This can be detailed as follows.

For the decoupled learning, the objectives are to

- 1) Propose new methods of decoupled learning that remove all the three BP lockings, thereby obtaining more promising acceleration performance compared with existing counterparts. In addition, the proposed methods must give comparable generalization performance to that of the conventional BP. That is, no obvious decrease in generalization performance should appear.
- 2) Propose methods to mitigate the gradient staleness issue commonly observed in decoupled learning. To the best of our knowledge, no prior arts in the area of decoupled learning have considered such mitigation yet.

For analytic learning, the objectives are to

- 1) Propose new analytic learning methods for multilayer networks, which avoid the aforementioned limitations (e.g., structural constraints and needing invertible

activation function).

- 2) Propose new methods for addressing the memory exploding issue in network learning caused by implementing the MP inverse in a batch-wise manner. The proposed methods should resemble the mini-batch SGD from the BP learning, which can process arbitrary number of samples in each iteration.

1.3 Major Contribution of the Thesis

In this thesis, the major contribution made to the areas of decoupled learning and analytic learning is summarized as follows.

- (i) The contributions to decoupled learning are mainly in the DG-based methods:
 - 1) A fully decoupled training scheme using DGs (FDG) is proposed, which breaks all the three BP lockings. The FDG splits a neural network into multiple modules and trains them independently and asynchronously using different workers (e.g., GPUs). Also, a gradient shrinking process is introduced to reduce the gradient staleness caused by utilizing the DGs. Theoretical proofs are provided showing that the FDG can converge to critical points under certain conditions. Experiments are conducted by training deep CNNs to perform classification tasks on CIFAR-10, CIFAR-100, and ImageNet. These experiments show comparable or better results of the proposed method compared with the state-of-the-art methods in terms of generalization and acceleration. In addition, it is demonstrated that the FDG is able to train various networks, including extremely deep ones (>1000 layers), in a decoupled fashion.

1.3. MAJOR CONTRIBUTION OF THE THESIS

- 2) An accumulated decoupled learning (ADL) is proposed, which incorporates module-wise gradient accumulation in order to mitigate the gradient staleness. Unlike prior arts ignoring the gradient staleness, in ADL the staleness is quantified in such a way that its mitigation can be quantitatively visualized. As a new learning scheme, the proposed ADL is also theoretically validated to converge to critical points in spite of its asynchronism. Extensive experiments on CIFAR-10 and ImageNet datasets are conducted, demonstrating that ADL gives promising generalization results while the state-of-the-art methods experience reduced generalization or divergence. In addition, the ADL is shown to have the fastest training speed among the compared methods.

As new decoupled learning techniques, the two proposed methods have distinctive focuses. The proposed FDG seeks to fill the gap in the current decoupled learning literature by breaking all the three lockings simultaneously. The FDG maximizes the learning efficiency while maintaining the generalization performance. In comparison, the ADL is also locking-free, but its focus is more on mitigating the gradient staleness (with theoretical proofs and empirical evidence). The ADL may be treated as a further improved version of the FDG, with an extra focus on the staleness issue.

(ii) Contributions to analytic learning:

- 1) A correlation projection network (CPNet) is proposed for pattern classification. The CPNet consists of multiple 2-layer modules which determine their parameters analytically. A label encoding is introduced to insert label information into hidden layers, and thus facilitates a locally super-

vised learning in each of the 2-layer modules where the analytic learning is achieved through LS estimators. Another core feature of the CPNet is the label decoding process that allows each module to give its own prediction. Such a unique design of prediction is able to reduce a considerable amount of time spent in the search of network structures during cross validations. As an analytic learning, the proposed CPNet only requires one visit of the training set, and is hence significantly faster than the commonly used BP, as verified in the experiments.

- 2) Based on the partitioning of the MP inverse, a block-wise recursive MP inverse formulation (BRMP) is proposed for analytic network learning with low-memory property while preserving its training effectiveness. The BRMP is an equivalent formulation to its batch-wise counterpart since neither approximation nor assumption is made in the derivation process. A further exploration of this recursive method leads to a switching structure among three different scenarios. This structure also reveals that the well-known recursive least squares (RLS) is a special case of the proposed technique. Subsequently, the BRMP is applied to the training of radial basis function networks as well as MLPs. The experimental validation covers both regression and classification tasks.

In summary, the proposed CPNet is a new form of multilayer analytic learning, which is no longer constrained by the existing limitations, e.g., needing invertible function and structural constraints. The BRMP reformulates the batch-wise LS solution based on MP inverse into a block-wise recursive implementation. It can be effortlessly applied in analytic learning to further reduce the memory consumption without obvious compromise of the learning efficiency.

1.4 Organization of the Thesis

The rest of this thesis is organized as follows. Chapter 2 reviews the existing literature for decoupled learning and analytic learning. Chapters 3-4 detail the newly proposed decoupled learning methods, while Chapters 5-6 show the contributions related to analytic learning.

In Chapter 3, the FDG, a decoupled learning method that addresses all the three BP lockings simultaneously, is presented. This includes a fully decoupled asynchronous learning algorithm with a gradient shrinking (GS) process to mitigate the accuracy decrease caused by utilizing the DGs. Chapter 4 introduces the ADL. This form of decoupled learning emphasizes mitigation of gradient staleness through a module-wise gradient accumulation technique. The mitigation is observed to be effective both theoretically and empirically.

In Chapter 5, the CPNet, a new analytic learning technique in training multi-layer neural networks, is developed. This includes a label encoding process for target preparation, a correlation projection technique for feature extraction, and an LS estimator for analytical weight determination. Chapter 6 gives the BRMP, a block-wise recursive implementation of the LS solution based on the MP inverse. The derivation leads to a switching structure among three different scenarios. In particular, it is shown that the well-known RLS is a special case of the proposed BRMP.

Subsequently, Chapter 7 concludes this thesis, and offers several possibilities of further research into decoupled learning as well as analytic learning. Finally, Appendix A contains the proofs for the theorems and lemmas in the previous chapters.

Notation: In this thesis, a non-boldface character (e.g., a or A) denotes a scalar; a boldface lowercase character (e.g., $\mathbf{a}$) represents a column vector; a boldface uppercase character (e.g., $\mathbf{A}$) indicates a matrix; $\mathbf{a} \in \mathbb{R}^K$ denotes a real-number vector $\mathbf{a}$ of K

1.4. ORGANIZATION OF THE THESIS

dimensions, and $\mathbf{A} \in \mathbb{R}^{N \times K}$ represents a real-number matrix $\mathbf{A}$ of size $N \times K$. In addition, *unless stated otherwise, the notations in each chapter are independent.*

Chapter 2

Literature Review

In this chapter, a detailed review of prior works related to the topic of this thesis is given. This mainly includes discussing related works regarding decoupled learning in Section 2.1, and analytic learning in Section 2.2. The exploration of related works in decoupled learning focuses on the LEL-based methods and the DG-based methods. The review of literature for analytic learning mainly consists of neural network training methods using the LS-based analytical solutions, and methods related to recursive MP inverse aiming reduce the memory consumption for the corresponding analytic learning approaches. An overview is presented in Figure 2.1.

2.1 Decoupled Learning

The BP is computationally inefficient due to its synchronous nature with the synchronism characterized by the forward, the backward and the update lockings [11]. In order to improve the efficiency, there have been a number of works in the name of decoupled learning on decoupling the training by splitting the network into multiple modules to facilitate inter-layer/inter-module model parallelization. The term

Figure 2.1: Diagram for an overview of the literature review.

decoupled in this area indicates that one or more of the BP lockings among modules are removed. In terms of parallelization, yet the decoupled learning has several benefits over methods based on the data parallelism or mixed-parallel paradigm [7, 42]. For instance, it does not reduce the overall batch size in each worker such that the performance of certain modules, such as Batch Normalization [43], can be maintained. In addition, it is able to straightforwardly parallelize recurrent neural networks (RNNs) [11]. These decoupling techniques mainly include the LEL-based methods, and the DG-based methods. In this thesis, the focus of decoupled learning lands on the DG-based methods.

2.1.1 Local Error Learning Based Methods

The LEL-based methods originated from the decoupled neural interface (DNI) [11], where the BP lockings were first defined. To overcome these lockings, the DNI at-

2.1. DECOUPLED LEARNING

tached each layer an auxiliary neural network for generating synthetic gradients to replace the real ones. The idea of using auxiliary networks to aid the learning was originally prototyped in the GoogLeNet [44] with the purpose of easing the gradient vanishing issue due to a deeper structure. Although the DNI is an effective tool for improving the BP efficiency, its generalization performance is shown to decrease severely and could encounter divergence in training very deep neural networks [13, 14]. In [45], the depth problem is resolved by attaching a regular CNN structure as the auxiliary network for training DNNs. Unlike the DNI, the auxiliary network is directly linked to the label which generates real but local gradients, allowing the lower modules to operate freely without waiting for the gradients passed from the upper modules. However, the generalization performance could not match the BP. Subsequently, the decoupled greedy learning (DGL) [46] improves the auxiliary CNN structure and achieves reasonably close generalization performance to the BP baselines. A local classifier with cross-entropy loss [47] is adopted to generate gradients for hidden layers. However, it has been shown that the local classifier alone fails to match the performance of a standard BP. In [31], a similarity measure combined with the local classifier (called pred-sim) is introduced to provide local error gradients. The mixed loss functions yield classification performance comparable to or even better than the BP baselines. However, this pred-sim method is only validated in VGG-like [9] networks (≤ 13 layers).

A common feature among these LEL techniques is the introduction of extra trainable parameters imposed by the auxiliary networks, which could sometimes be rather expensive. For instance, to match the standard BP, the local learning in [31] needs to train several times more parameters.

2.1.2 Delayed Gradient Based Methods

The DG-based decoupled learning adopts DGs (a form of older gradients) to achieve asynchronism among different modules, which accordingly breaks the BP lockings. In general, the DG-based methods deliver much more promising generalization performance than that produced by the LEL-based methods. This branch of decoupled learning is very new and relatively immature, and is mainly applied in image recognition tasks. In [13], the DDG parallelizes the backward pass among modules after conducting a full forward pass, achieving decoupled learning by breaking the backward locking. The DDG can train very deep (up to 110 layers) CNNs, and shows no performance loss in certain architectures while reducing the training time. Since the DDG is still constrained by the forward locking, the acceleration is relatively limited even with multiple workers. The FR [14] following the DDG also breaks the backward locking. Differently, it adopts the recomputation technique for memory reduction, and has been shown to even outperform BP for several deep architectures. However, the FR introduces additional computation and is therefore even slower than the DDG. Recently, the DG-based methods has also been introduced in the training of video models [48].

The existing DG-based methods have not yet fulfilled their acceleration potentials. Suppose that K workers are used, efficient acceleration methods should obtain a speedup approaching K times compared with the one using the original BP. This is not the case among the current DG-based methods. The accelerations of the DDG and the FR are underdeveloped as they are constrained by the other two lockings (i.e., the forward and the update lockings). It is shown in Chapter 3 that, upon removing all the three lockings, the proposed FDG can achieve an ideal K -time acceleration with the empirical results approaching this limit.

Besides the not-yet-fulfilled acceleration ability, another concern of the DG-based methods is the gradient staleness caused by utilizing older gradients. The staleness may lead to decrease in generalization or even divergence. To detail this issue, for convenience, the partition size (PS) is referred to as the number of modules being partitioned from a network. A large number of partitioned modules (i.e., high PS) would encourage a high degree of efficiency being improved. However, it also indicates that the gradient staleness grows accordingly, imposing stronger impact on the network learning. To the best of our knowledge, there are no existing works of decoupled learning involving the mitigation of such staleness issue. In addition, the current methods are only validated using very small PS values (e.g., ≤ 4 [13]). It is shown in Chapter 4 that the gradient staleness can be effectively mitigated by the proposed ADL, such that the decoupled learning becomes more accurate and stable, even with a large PS.

2.1.3 Other Works Related to Decoupled Learning

There are also related works for accelerating the training of DNNs which share similarities to the decoupled learning with either removing the global BP or adopting DGs. One is to promote *biological plausibility* [49–51], which focuses on removing the weight symmetry and possible gradient propagation from the BP procedure. The feedback alignment (FA) [49] removes the weight symmetry by replacing symmetrical weights with random ones during the backward pass. Direct feedback alignment [50] following the FA replaces the BP with a random projection and potentially facilitates a simultaneous training for all layers during the backward pass. However, these biologically inspired approaches tend to suffer from performance loss, and are shown to scale poorly on more complex datasets [52]. Another set of methods, named asynchronous

stochastic gradient descent (ASGD) [53–55], involve DGs to perform asynchronous parallelization. The major difference from the decoupled learning is that the ASGD essentially belongs to the data parallelism category as each work handles a complete network replica. In addition, pipeline BP share certain similarity to the decoupled learning. Gpipe proposed in [56] splits a network into several modules and breaks a mini-batch into several micro-batches to conduct pipeline model parallelization. Unlike the decoupled learning, the Gpipe is a form of synchronous model parallelism, which does not involve DGs.

2.2 Analytic Learning with Moore-Penrose Inverse

Here, the review of analytic learning starts at introducing the MP inverse [33], which is the key component in this research area. Then the existing analytic learning methods are reviewed, including the shallow network learning and the multilayer network learning. Finally, the related works are given regarding the recursive implementation of the MP inverse.

2.2.1 Analytic Learning Methods

Conventional analytic learning started with shallow networks [23–28]. A good example is the well-known radial basis function (RBF) network [22]. It learns its parameters in the second layer using a closed-form LS solution after conducting a feature transformation in the first layer. Investigation in random-weighted networks [29, 30] suggests that selection of weights in the first layer is relatively unimportant where a random assignment of weights is good enough for competing performance. This observation has been supported by extensive experiments on such networks with randomly projected weights that optimize a classification error based objective function [57]. Guo

et al [58] adopts a similar randomization technique for the hidden layer weights with a *Tanh* activation function to train a shallow network with multiple outputs. The idea of adopting random weights in the hidden layer is also seen in [59,60] in the name of extreme learning machines (ELMs). The most evident benefit of analytic learning for training neural networks is the fast single-epoch learning for training data of small to medium sizes.

The progress of analytic learning over recent years has shown an increasing attention towards incorporation of the fast-learning attribute to deep networks. The challenge of such development mainly lies in the learning of hidden layers. One possible solution is to feed the label information into hidden layers to facilitate locally supervised learning. In [18], the KARnet has been proposed to analytically solve multilayer network learning with a sequential label projection into the hidden layers. The projection is based on a series of MP inverse operations conditioned on an invertible activation function. The KARnet has been observed to give good performance on datasets of small to medium sizes. The multilayer ELM proposed in [32] also needs an invertible activation function, but shows limited improvement over the shallow networks. In [61], the pseudoinverse autoencoder (PILAE) introduces a stacking multilayer autoencoder using the input as label for hidden layers. There are also stacking-based network learning which have achieved analytic learning, such as [62–64].

Apart from inserting the label information, there are also other options to address analytic learning in hidden layers. In [17], the PIL trains the neural networks through adding extra layers to eliminate the training errors. The PIL determines the network weights using the MP inverse of a feature matrix, which enforces the hidden layer size to be matched with the size of the given training set. This could lead to tremendous

memory consumption for large datasets. A method called ANnet [65] trains a network by treating the normalized transpose of a feature matrix as its hidden weights. This method demonstrates promising performance on several datasets, but it potentially suffers from the memory problem similar to that in the PIL. The FF technique [21] projects the label information via a k-nearest neighbors clustering technique, and utilizes the PCA for preprocessing. The PCA module improves the feature extraction process of the FF method but results in extra complexity and memory consumption.

In summary, existing attempts of analytic learning either require invertible activation functions (e.g., the KARnet and multilayer ELM) or have strong structural constraints (e.g., the PIL and ANnet) for multilayer network learning. In Chapter 5, a CPNet for multilayer analytic network learning is proposed, which is free from the aforementioned limitations.

2.2.2 MP Inverse Decomposition and Recursive MP Inverse Learning

In general, analytic learning methods train neural networks in a batch-wise manner. That is, they need to process the entire dataset in a single computation, which has a memory demand proportional to the size of the dataset. This likely leads to OOM for a large dataset.

Decomposition of the MP inverse is an important step to circumvent the memory problem. More specifically, decomposing the MP inverse can reproduce the LS solution while avoiding the memory-intensive batch-wise learning. The most common decomposition scheme is the recursive MP inverse (RMP) through calculating the matrix's MP inverse one row/column at a time [41]. In [39], the RMP is adopted for a reformulation of the batch-wise LS, leading to an exact recursive solution that

gradually updates its parameters by taking one linear equation per iteration. Based on the RMP, [40] is able to train the shallow linear neural networks as well as the shallow nonlinear neural networks through the Taylor's expansion. Recently, a method called LM-KARnet [19] adopting the RMP-like technique has been formulated for sample-wise MLP training, which allows a small amount of memory usage.

In spite of being memory-efficient, the above sample-wise methods could suffer serious decrease of speed especially when the size of a dataset is large. To improve the situation capitalizing on the parallel processing capability, a block-wise MP inverse that processes multiple rows/columns at each time has been crafted in [66]. The block-wise decomposition splits an MP inverse into calculations involving two arbitrary submatrices. However, the block-wise decomposition is much more complicated than the RMP such that an extension to the LS solution is not a trivial task. The goal in this part of analytic learning is to develop a block-wise recursive method for network learning by an exact formulation of the MP inverse. The developed technique is to be equally memory-efficient but significantly faster than the sample-wise learning methods. In Chapter 6, the proposed BRMP achieves a recursive implementation of LS solution based on the block-wise recursive MP inverse, which is shown to exactly reproduce the computation of the batch-wise solution without imposing any condition on the data matrix.

Chapter 3

Fully Decoupled Neural Network Learning Using Delayed Gradients

It has been shown in Section 1.1.1 and Section 2.1 that the decoupled learning can roughly be categorized into two classes: the LEL-based (e.g., DNI [11] and DGL [46]) and DG-based (e.g., DDG [13] and FR [14]) methods. In comparison, the LEL-based methods fully decouple (i.e., breaking all the BP lockings) the network learning but introduce extra trainable parameters and could experience nontrivial decline of generalization performance. The existing DG-based methods usually obtain good generalization performance but breaks the backward locking only. This has motivated us to develop a decoupled learning technique which shares both merits from these two classes of methods. In this chapter, a fully decoupled learning method using delayed gradients (FDG) is proposed, which breaks all the BP lockings without introducing extra trainable parameters (see Table 3.1).

The rest of this chapter is presented as follows. Section 3.1 gives the background knowledge to develop the FDG. Section 3.2 elaborates the algorithmic details of the

3.1. PRELIMINARIES

Table 3.1: Comparison with state-of-the-art methods in terms of lockings and auxiliary networks.

Methods	DDG [13]	FR [14]	DNI [11]	DGL [46]	FDG (ours)
Lockings	Yes	Yes	No	No	No
Auxiliary networks	No	No	Yes	Yes	No

Table 3.2: Table of Notations.

Notations	Description
$\mathbf{z}_l^t$	Activation vector at layer l of batch t .
$\boldsymbol{\theta}_l^t$	Parameter vector at layer l of batch t .
$\bar{\mathbf{g}}_{\boldsymbol{\theta}_l}^t$	Gradient vector of $\boldsymbol{\theta}_l$ calculated using gradient descent of batch t .
$\mathbf{g}_{\boldsymbol{\theta}_l}^t$	Gradient vector of $\boldsymbol{\theta}_l$ calculated using stochastic gradient descent.
$\mathbf{g}_{\mathbf{z}_l}^t$	Gradient vector of $\mathbf{z}_l$ of batch t .
$f_{\mathbf{x}_t}(\boldsymbol{\theta}^t)$	loss function w.r.t. data batch $\mathbf{x}_t$.
$\hat{\mathbf{g}}$	Estimated gradient

proposed method. The convergence analysis is given in Section 3.3. Experimental validation is presented in Section 3.4. The chapter is then summarized in Section 3.5.

3.1 Preliminaries

Here, some background knowledge for training a feedforward neural network is revisited. The forward, backward, and update lockings [11] are also explained. For convenience, the frequently used notations are presented in Table 3.2.

Assume that we need to train an L -layer network. The l^{th} ($1 \leq l \leq \mathcal{L}$) layer produces an activation $\mathbf{z}_l = F_l(\mathbf{z}_{l-1}; \boldsymbol{\theta}_l)$ by taking $\mathbf{z}_{l-1}$ as its input, where F_l is an activation function and $\boldsymbol{\theta}_l \in \mathbb{R}^{n_l}$ is a column vector representing the weights in layer l . The sequential generation of the activations leads to the *forward locking* since $\mathbf{z}_l$ will not be available before all the dependent activations are obtained. Let $\boldsymbol{\theta} = [\boldsymbol{\theta}_1^T, \boldsymbol{\theta}_2^T, \dots, \boldsymbol{\theta}_{\mathcal{L}}^T]^T \in \mathbb{R}^{\sum_{i=1}^L n_i}$ denote the parameter vector for the whole network.

3.1. PRELIMINARIES

Assume f is a loss function that maps a high-dimensional vector to a scalar. The learning of the feedforward network can then be summarized as the following optimization problem:

$$\underset{\boldsymbol{\theta}}{\text{minimize}} \quad f_{\mathbf{x}}(\boldsymbol{\theta}) \quad (3.1)$$

where $\mathbf{x}$ represents the entire input-label information (or the entire dataset). In the rest of this thesis, $f(\boldsymbol{\theta})$ is used to represent $f_{\mathbf{x}}(\boldsymbol{\theta})$ for convenience.

The gradient descent algorithm is often used to solve (3.1) by updating the parameter $\boldsymbol{\theta}$ iteratively. At iteration t , we have

$$\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^t - \gamma_t \bar{\mathbf{g}}_{\boldsymbol{\theta}}^t. \quad (3.2)$$

Or equivalently,

$$\boldsymbol{\theta}_l^{t+1} = \boldsymbol{\theta}_l^t - \gamma_t \bar{\mathbf{g}}_{\boldsymbol{\theta}_l}^t, \quad l = 1, \dots, \mathcal{L} \quad (3.3)$$

where γ_t is the learning rate at iteration t , and $\bar{\mathbf{g}}_{\boldsymbol{\theta}}^t = [(\bar{\mathbf{g}}_{\boldsymbol{\theta}_1}^t)^T, (\bar{\mathbf{g}}_{\boldsymbol{\theta}_2}^t)^T, \dots, (\bar{\mathbf{g}}_{\boldsymbol{\theta}_L}^t)^T]^T \in \mathbb{R}^{\sum_{i=1}^L n_i}$ is the gradient vector obtained by

$$\bar{\mathbf{g}}_{\boldsymbol{\theta}_l}^t = \frac{\partial f(\boldsymbol{\theta}^t)}{\partial \boldsymbol{\theta}_l^t}. \quad (3.4)$$

Note that notation t here is also referred to as the *batch index* indicating the t^{th} batch being processed. In the synchronous BP case, the processing of the t^{th} data batch happens at iteration t , so their names (i.e., iteration and batch index) can be used interchangeably. However, this might not hold in the asynchronous learning such as the decoupled learning. That is, at iteration t_1 , the network module could be

3.1. PRELIMINARIES

processing the t_2 data batch with $t_2 \neq t_1$ (see Section 3.2 for examples).

If the training sample size is large, the SGD is often used as a replacement such that

$$\mathbf{g}_{\boldsymbol{\theta}_l}^t = \frac{\partial f_{\mathbf{x}_t}(\boldsymbol{\theta}^t)}{\partial \boldsymbol{\theta}_l^t} \quad (3.5)$$

where $\mathbf{x}_t$ is a mini-batch of $\mathbf{x}$. Note that the bar “ $-$ ” has been removed to indicate the difference from (3.4). Thus the parameter can be updated through

$$\boldsymbol{\theta}_l^{t+1} = \boldsymbol{\theta}_l^t - \gamma_t \mathbf{g}_{\boldsymbol{\theta}_l}^t, \quad l = 1, \dots, \mathcal{L}. \quad (3.6)$$

Assume that each sample is randomly drawn from the dataset with a uniform distribution. Then the gradient is unbiased:

$$\mathbb{E}_{\mathbf{x}_t} \{\mathbf{g}_{\boldsymbol{\theta}_l}^t\} = \bar{\mathbf{g}}_{\boldsymbol{\theta}_l}^t \quad (3.7)$$

where the expectation $\mathbb{E}_{\mathbf{x}_t}$ is taken w.r.t. the random variable that draws $\mathbf{x}_t$ from the dataset.

To obtain the gradient vectors, the BP can be employed. One could calculate the gradients in layer l using the gradients back-propagated from layers j and i ($l < j < i$):

$$\mathbf{g}_{\boldsymbol{\theta}_l}^t = \frac{\partial f_{\mathbf{x}_t}(\boldsymbol{\theta}^t)}{\partial \boldsymbol{\theta}_l^t} = \frac{\partial \mathbf{z}_j^t}{\partial \boldsymbol{\theta}_l^t} \frac{\partial f_{\mathbf{x}_t}(\boldsymbol{\theta}^t)}{\partial \mathbf{z}_j^t} = \frac{\partial \mathbf{z}_j^t}{\partial \boldsymbol{\theta}_l^t} \mathbf{g}_{\mathbf{z}_j}^t \quad (3.8)$$

where

$$\mathbf{g}_{\mathbf{z}_j}^t = \frac{\partial f_{\mathbf{x}_t}(\boldsymbol{\theta}^t)}{\partial \mathbf{z}_j^t} = \frac{\partial \mathbf{z}_i^t}{\partial \mathbf{z}_j^t} \frac{\partial f_{\mathbf{x}_t}(\boldsymbol{\theta}^t)}{\partial \mathbf{z}_i^t} = \frac{\partial \mathbf{z}_i^t}{\partial \mathbf{z}_j^t} \mathbf{g}_{\mathbf{z}_i}^t. \quad (3.9)$$

3.2. FULLY DECOUPLED NEURAL NETWORK LEARNING

Note that $\mathbf{g}_{\mathbf{z}_j}^t$ —the gradient vector w.r.t. activation $\mathbf{z}_j$ —is introduced because it travels between modules as an important part of the proposed FDG. (3.8) and (3.9) indicate that $\mathbf{g}_{\boldsymbol{\theta}_l}^t$ depends on $\mathbf{g}_{\mathbf{z}_j}^t$ and $\mathbf{g}_{\mathbf{z}_i}^t$. In other words, the gradient in layer l remains unavailable until all the dependent gradients are obtained. This is also known as the *backward locking*. In addition, the parameter update is not permitted before all layers complete executing their forward passes. This is recognized as the *update locking*. In the following, a full decoupling (i.e., the forward, backward and update unlockings) is shown to be achievable.

3.2 Fully Decoupled Neural Network Learning

In this section, the details of the proposed FDG are given. This technique consists a fully decoupled asynchronous learning algorithm with a GS process to mitigate the accuracy decrease caused by the DGs.

3.2.1 The Proposed FDG

The network is first partitioned into K modules with each module containing a stack of layers. Accordingly, the set of the layer indices $\{1, \dots, \mathcal{L}\}$ is split into $\{q(1), q(2), \dots, q(K)\}$ where $q(k) = \{m_k, m_k + 1, \dots, m_{k+1} - 1\}$ denotes the layer indices in module k .

As illustrated in Figure 3.1(a), during the decoupled learning, module k performs a forward pass and a backward pass in each iteration. The forward pass is executed based on the delayed activation previously passed from module $k - 1$ while the backward pass is conducted using the DG previously sent by module $k + 1$. Note that all the modules are executed with the same procedures simultaneously, hence achieving the parallel training of different modules. After executing both passes, the gradient

3.2. FULLY DECOUPLED NEURAL NETWORK LEARNING

Figure 3.1: Illustration of the proposed FDG: a $K = 3$ example. (a) By delaying the activations and gradients, the FDG allows the modules (in different colors) to be trained asynchronously in parallel. The communications among modules happen right before the iteration ends. It is observed that there is a delay of $2(K - k)$ of gradients in module k with a split of K . For instance, for $K = 3$, module $k = 1$ (gray) generates an activation at iteration 1 but the gradient of this batch arrives at iteration 5. (b) In the FDG, the backward pass (step B) is executed in the previously saved computation graph while the forward pass (step A) happens in the current one. This is how the FDG overcomes the sequential nature of the standard BP.

of the module input is passed to module $k - 1$, while the module output is sent to module $k + 1$ as its new input. This can be detailed by the following steps for module k ($1 < k < K$):

Forward Pass

At iteration t , the input $\mathbf{z}_{m_k-1}^{t-k+1}$ (previously sent by module $k - 1$) is fed into module k , which in turn gives a module output (activation) $\mathbf{z}_{m_{k+1}-1}^{t-k+1}$. Notation $t - k + 1$ instead of t is adopted because at iteration t the delayed activation generated by batch $t - k + 1$ (see Figure 3.1(a)) is used.

Backward Pass

At iteration t , the DG $\mathbf{g}_{\mathbf{z}_{m_{k+1}-1}}^{t-2K+k+1}$ previously received from module $k + 1$ is utilized to resume the BP procedure. The superscript $t - 2K + k + 1$ is adopted because there is a delay of $2(K - k)$ of gradients (see Figure 3.1(a)) w.r.t. the forward pass that uses

3.2. FULLY DECOUPLED NEURAL NETWORK LEARNING

batch $t - k + 1$. Thus, for each layer ($m_k \leq l \leq m_{k+1} - 1$), the gradient is obtained by:

$$\hat{\mathbf{g}}_{\boldsymbol{\theta}_l}^{t-k+1} = \frac{\partial \mathbf{z}_{m_{k+1}-1}^{t-2K+k+1}}{\partial \boldsymbol{\theta}_l^{t-2K+k+1}} \mathbf{g}_{\mathbf{z}_{m_{k+1}-1}}^{t-2K+k+1}. \quad (3.10)$$

Note that (3.10) must utilize the delayed activation of batch $t - 2K + k + 1$. It is only reasonable to calculate the gradient based on the activation generated by the same training batch. Subsequently, the module can be updated through

$$\boldsymbol{\theta}_l^{t-k+2} = \boldsymbol{\theta}_l^{t-k+1} - \gamma_{t-k+1} \hat{\mathbf{g}}_{\boldsymbol{\theta}_l}^{t-k+1}. \quad (3.11)$$

After completing the BP in this module, $\mathbf{g}_{\mathbf{z}_{m_k-1}}^{t-2K+k+1}$, the gradient of the module input, is saved for communication.

Subsequently, at iteration t , send $\mathbf{g}_{\mathbf{z}_{m_k-1}}^{t-2K+k+1}$ to module $k - 1$ and pass $\mathbf{z}_{m_{k+1}-1}^{t-k+1}$ to module $k + 1$ as its new input. Note that the first ($k = 1$) and the last ($k = K$) modules are executed similarly but require less work in the communication step. The first module does not need to send the gradient as there is no lower module, and the last module receives its gradient propagated directly from the loss function.

The aforementioned forward, backward, and communication steps break all the lockings in [11]. Firstly, the global BP is broken into module-wise BP running in parallel, which achieves the *backward unlocking*. Secondly, each of the split modules processes the training data from different batches, leading to an asynchronous module parallelization, hence the *forward unlocking*. Finally, each module can be immediately updated without waiting for other modules to complete their forward passes, so the *update unlocking* is also achieved.

Note that $t - k + 1$ instead of t is adopted in the update formula (3.11) in corre-

3.2. FULLY DECOUPLED NEURAL NETWORK LEARNING

spondence to the activation index (e.g., $\mathbf{z}_{m_{k+1}-1}^{t-k+1}$) at iteration t . One could straightforwardly see that, for module k at iteration $t + k - 1$, (3.11) can be equivalently shifted to

$$\boldsymbol{\theta}_l^{t+1} = \boldsymbol{\theta}_l^t - \gamma_t \hat{\mathbf{g}}_{\boldsymbol{\theta}_l}^t \quad (3.12)$$

with

$$\hat{\mathbf{g}}_{\boldsymbol{\theta}_l}^t = \frac{\partial \mathbf{z}_{m_{k+1}-1}^{t-2K+2k}}{\partial \boldsymbol{\theta}_l^{t-2K+2k}} \mathbf{g}_{\mathbf{z}_{m_{k+1}-1}^{t-2K+2k}}. \quad (3.13)$$

Adopting (3.12) over (3.11) would benefit the subsequent convergence analysis in the next section. Let $d_{k,t} = t - 2(K - k)$, $\hat{\mathbf{g}}_{\boldsymbol{\theta}_l}^t$ can be further unpacked such that

$$\hat{\mathbf{g}}_{\boldsymbol{\theta}_l}^t = \frac{\partial \mathbf{z}_{m_{k+1}-1}^{d_{k,t}}}{\partial \boldsymbol{\theta}_l^{d_{k,t}}} \frac{\partial f_{\mathbf{x}_{d_{k,t}}}(\boldsymbol{\theta}^{d_{k,t}})}{\partial \mathbf{z}_{m_{k+1}-1}^{d_{k,t}}} = \frac{\partial f_{\mathbf{x}_{d_{k,t}}}(\boldsymbol{\theta}^{d_{k,t}})}{\partial \boldsymbol{\theta}_l^{d_{k,t}}} = \mathbf{g}_{\boldsymbol{\theta}_l}^{d_{k,t}} \quad (3.14)$$

and (3.12) can be rewritten as

$$\boldsymbol{\theta}_l^{t+1} = \boldsymbol{\theta}_l^t - \gamma_t \mathbf{g}_{\boldsymbol{\theta}_l}^{d_{k,t}} \quad (3.15)$$

which is observed to take DGs with a delay of $2(K - k)$ compared with (3.6).

Analysis of Speedup: Assume that a network is evenly split into K modules in terms of computing workload. In the ideal case where other time consumptions such as communication are excluded, Table 3.3 shows that a linear (K -time) speedup can be achieved with the FDG. This is the highest among all the decoupling methods.

3.2. FULLY DECOUPLED NEURAL NETWORK LEARNING

Table 3.3: Ideal speedup of different decoupling methods for a network evenly split into K modules. $\mathcal{T}_f$, $\mathcal{T}_b$ and $\mathcal{T}_{aux}$ denote the computation time executing the forward, the backward pass, and the auxiliary network, respectively.

Methods	BP [67]	DDG [13]	FR [14]	DNI [11] & DGL [46]	FDG (ours)
Time	$\mathcal{T}_f + \mathcal{T}_b$	$\mathcal{T}_f + \frac{\mathcal{T}_b}{K}$	$\mathcal{T}_f + \frac{\mathcal{T}_f + \mathcal{T}_b}{K}$	$\frac{\mathcal{T}_f + \mathcal{T}_b}{K} + \mathcal{T}_{aux}$	$\frac{\mathcal{T}_f + \mathcal{T}_b}{K}$

3.2.2 The Gradient Shrinking Process

Using the DGs enables inter-layer model parallelization but could also lead to certain performance decrease. This is a common phenomenon observed in algorithms with stale gradients [55]. To compensate such performance loss, a GS process is introduced before back-propagating the DGs through each module.

The GS process works in a direct manner. At iteration t , before executing the local BP in module k , the gradient is shrunk by being multiplied with a shrinking factor β ($0 < \beta \leq 1$). This can be shown by modifying (3.13) as

$$\hat{\mathbf{g}}_{\theta_l}^t = \beta \frac{\partial \mathbf{z}_{m_{k+1}-1}^{d_{k,t}}}{\partial \theta_l^{d_{k,t}}} \hat{\mathbf{g}}_{\mathbf{z}_{m_{k+1}-1}}^{d_{k,t}}. \quad (3.16)$$

The module is then updated through (3.12). Note that unlike (3.13) where the received gradient is denoted by $\mathbf{g}_{\mathbf{z}_{m_{k+1}-1}}^{d_{k,t}}$, the received gradient in (3.16) is represented by $\hat{\mathbf{g}}_{\mathbf{z}_{m_{k+1}-1}}^{d_{k,t}}$. This is because the gradient in module k is affected by multiple GS processes starting from the top module where the gradient is not shrunk. This can

3.2. FULLY DECOUPLED NEURAL NETWORK LEARNING

Figure 3.2: An intuitive interpretation for the GS process. (a) The DGs could cause the learning to miss the local minimum to a large margin. (b) By reducing the gradient strength, the learning could be brought closer to the local minimum than that without the GS process.

be illustrated by unpacking (3.16):

$$\begin{aligned}
 \hat{\mathbf{g}}_{\theta_l}^t &= \beta^2 \frac{\partial \mathbf{z}_{m_{k+1}-1}^{d_{k,t}}}{\partial \theta_l^{d_{k,t}}} \frac{\partial \mathbf{z}_{m_{k+2}-1}^{d_{k,t}}}{\partial \mathbf{z}_{m_{k+1}-1}^{d_{k,t}}} \mathbf{g}_{\mathbf{z}_{m_{k+2}-1}}^{d_{k,t}} \\
 &\vdots \\
 &= \beta^{K-k} \frac{\partial \mathbf{z}_{m_{k+1}-1}^{d_{k,t}}}{\partial \theta_l^{d_{k,t}}} \cdots \frac{\partial \mathbf{z}_{m_K-1}^{d_{k,t}}}{\partial \mathbf{z}_{m_{K-1}-1}^{d_{k,t}}} \mathbf{g}_{\mathbf{z}_{m_K-1}}^{d_{k,t}} \\
 &= \beta^{K-k} \mathbf{g}_{\theta_l}^{d_{k,t}}
 \end{aligned} \tag{3.17}$$

which shrinks the gradient by a factor of β^{K-k} in module k . Note that if $\beta = 1$, the GS process is not used as $\beta^{K-k} = 1$.

The GS process can be interpreted in an intuitive way shown in Figure 3.2. The DGs, especially with longer delays, would lead to deteriorated performance [55]. Figure 3.2(a) shows a scenario where the DGs cause the learning to miss the local minimum to a large margin. By using the shrunk DGs, there is a better chance of reducing the gradient staleness (see Figure 3.2(b) for an illustration). The GS process works

3.2. FULLY DECOUPLED NEURAL NETWORK LEARNING

similarly to scaling down the learning rate, but they are not necessarily identical when certain regularization techniques are involved. Take the widely used *weight decay* [68] as an example. Involving such a technique, the update rule in (3.15) is modified as

$$\boldsymbol{\theta}_l^{t+1} = \boldsymbol{\theta}_l^t - \gamma_t \mathbf{g}_{\boldsymbol{\theta}_l}^{d_{k,t}} - \gamma_t \lambda \boldsymbol{\theta}_l^t \quad (3.18)$$

where λ is the decay factor usually taking a small value. The GS process, i.e., $\mathbf{g}_{\boldsymbol{\theta}_l}^{d_{k,t}} \rightarrow \beta^{K-k} \mathbf{g}_{\boldsymbol{\theta}_l}^{d_{k,t}}$, rewrites (3.18) into $\boldsymbol{\theta}_l^{t+1} = \boldsymbol{\theta}_l^t - \gamma_t \beta^{K-k} \mathbf{g}_{\boldsymbol{\theta}_l}^{d_{k,t}} - \gamma_t \lambda \boldsymbol{\theta}_l^t$. This differs from reducing the learning rate correspondingly, i.e., $\gamma_t \rightarrow \beta^{K-k} \gamma_t$, which rewrites (3.18) into $\boldsymbol{\theta}_l^{t+1} = \boldsymbol{\theta}_l^t - \gamma_t \beta^{K-k} \mathbf{g}_{\boldsymbol{\theta}_l}^{d_{k,t}} - \gamma_t \beta^{K-k} \lambda \boldsymbol{\theta}_l^t$. To mitigate the gradient staleness, the GS process rather than scaling the learning rate is adopted for the following reasons. The GS process allows us to appreciate the stale gradient mitigation more directly and intuitively. That is, how much the calculated gradient is trusted. The learning rate scaling tends to also rescale other factors (e.g., weight decay) apart from the gradients, which complicates the interpretation. In addition, the empirical results (see Figure 3.3(b)) show that, in general, adopting the GS process leads to some performance gain in comparison. The proposed FDG with SGD optimizer is summarized in Algorithm 3.1.

Comparison among Methods Adopting Delayed Gradients: Although DGs are adopted like in the DDG [13], FR [14], and ASGD methods [55], the FDG is different from these techniques in several ways. The DDG and FR only break the backward locking, but the FDG is able to break all the lockings, leading to a more efficient training. Unlike the ASGD-based methods which let each worker handle the entire network, the FDG trains the network by splitting it into modules where each module is handled by one specific worker.

3.3. CONVERGENCE ANALYSIS

Algorithm 3.1 FDG with SGD

Partition the network into K modules, and set learning rate γ_t , gradient shrinking factor β ;

for each iteration **do**

for $k \leftarrow 1$ to K (**Parallel**) **do**

Forward pass: execute the forward pass to generate the module output $\mathbf{z}_{m_{k+1}-1}^{t-k+1}$;

Backward pass: compute the shrunk DG in each layer $\hat{\mathbf{g}}_{\theta_i}^{t-k+1}$ (3.16), and gradient of the module input $\hat{\mathbf{g}}_{\mathbf{z}_{m_k-1}}^{t-2K+k+1}$;

Update: update the module through (3.11);

Communication: pass $\mathbf{z}_{m_{k+1}-1}^{t-k+1}$ to module $k+1$ as its input, and send $\hat{\mathbf{g}}_{\mathbf{z}_{m_k-1}}^{t-2K+k+1}$ to module $k-1$;

end

end

3.3 Convergence Analysis

In this section, the convergence analysis is given for the proposed FDG. It is shown that the proposed method can converge to critical points under certain conditions.

Prior to the analysis, $\boldsymbol{\theta}^t, \mathbf{g}_{\boldsymbol{\theta}}^t$ and $\bar{\mathbf{g}}_{\boldsymbol{\theta}}^t$ are written in terms of modules as follows:

$$\begin{aligned}\boldsymbol{\theta}^t &= [(\boldsymbol{\theta}_{q(1)}^t)^T, \dots, (\boldsymbol{\theta}_{q(K)}^t)^T]^T, & \boldsymbol{\theta}_{q(k)}^t &= [(\boldsymbol{\theta}_{m_k}^t)^T, \dots, (\boldsymbol{\theta}_{m_{k+1}-1}^t)^T]^T \\ \mathbf{g}_{\boldsymbol{\theta}}^t &= [(\mathbf{g}_{\boldsymbol{\theta}_{q(1)}}^t)^T, \dots, (\mathbf{g}_{\boldsymbol{\theta}_{q(K)}}^t)^T]^T, & \mathbf{g}_{\boldsymbol{\theta}_{q(k)}}^t &= [(\mathbf{g}_{\boldsymbol{\theta}_{m_k}}^t)^T, \dots, (\mathbf{g}_{\boldsymbol{\theta}_{m_{k+1}-1}}^t)^T]^T \\ \bar{\mathbf{g}}_{\boldsymbol{\theta}}^t &= [(\bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(1)}}^t)^T, \dots, (\bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(K)}}^t)^T]^T, & \bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^t &= [(\bar{\mathbf{g}}_{\boldsymbol{\theta}_{m_k}}^t)^T, \dots, (\bar{\mathbf{g}}_{\boldsymbol{\theta}_{m_{k+1}-1}}^t)^T]^T.\end{aligned}$$

The convergence analysis is based on the below assumptions.

Assumption 3.3.1. *Lipschitz continuity of gradients for loss functions $f(\boldsymbol{\theta})$ and $f_{\mathbf{x}_t}(\boldsymbol{\theta})$, which means $\exists L \in \mathbb{R}^+$ such that:*

$$\|\bar{\mathbf{g}}_{\boldsymbol{\theta}}^{t_1} - \bar{\mathbf{g}}_{\boldsymbol{\theta}}^{t_2}\|_2 \leq L \|\boldsymbol{\theta}^{t_1} - \boldsymbol{\theta}^{t_2}\|_2 \quad (3.19)$$

$$\|\mathbf{g}_{\boldsymbol{\theta}_{q(k)}}^{t_1} - \mathbf{g}_{\boldsymbol{\theta}_{q(k)}}^{t_2}\|_2 \leq L \|\boldsymbol{\theta}_{q(k)}^{t_1} - \boldsymbol{\theta}_{q(k)}^{t_2}\|_2 \quad (3.20)$$

3.3. CONVERGENCE ANALYSIS

where $\|\cdot\|_2$ is the l_2 -norm operator.

Assumption 3.3.2. *Bounded variance of the stochastic gradient, which means that $\forall t$, there exists a constant $A > 0$ such that:*

$$\|\mathbf{g}_\theta^t\|_2^2 \leq A. \quad (3.21)$$

These two assumptions are commonly adopted in convergence analysis [13, 69] in deep learning. Accordingly, FDG's convergence property can be obtained in the following theoretical findings.

Theorem 3.3.1. *Let Assumptions 3.3.1 and 3.3.2 hold. Assume that the learning rate is non-increasing and $L\gamma_t \leq 1$. The proposed FDG in Algorithm 3.1 satisfies*

$$\mathbb{E}_{\mathbf{x}_t}\{f(\boldsymbol{\theta}^{t+1})\} - f(\boldsymbol{\theta}^t) \leq -\frac{\gamma_t}{2}Z_1 + \gamma_t^2 Z_2 \quad (3.22)$$

where

$$Z_1 = \sum_{k=1}^K \beta^{K-k} \left\| \bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^t \right\|_2^2, \quad Z_2 = LA \sum_{k=1}^K \beta^{K-k} + LA \sum_{k=1}^K \beta^{3(K-k)} 2(K-k).$$

Proof. See Appendix A.1.1. □

Theorem 3.3.1 gives an important indication of the FDG's convergence behavior. That is, if the right hand side (RHS) of (3.22) is negative, i.e.,

$$-\frac{\gamma_t}{2}Z_1 + \gamma_t^2 Z_2 < 0 \Rightarrow \gamma_t < \min\{1/L, Z_1/2Z_2\},$$

the expected loss $\mathbb{E}_{\mathbf{x}_t}\{f(\boldsymbol{\theta}^{t+1})\}$ would decrease. The convergence evidence is given in the following theorems.

3.3. CONVERGENCE ANALYSIS

Theorem 3.3.2. *Assume that Assumptions 3.3.1 and 3.3.2 hold. The learning rate is non-increasing and satisfies $L\gamma_t \leq 1$. Let $\boldsymbol{\theta}^*$ be the global minimizer and $\mathcal{T}_T = \sum_{t=0}^{T-1} \gamma_t$. Then*

$$\frac{1}{\mathcal{T}_T} \sum_{t=0}^{T-1} \gamma_t \mathbb{E}\{\|\bar{\mathbf{g}}_{\boldsymbol{\theta}}^t\|_2^2\} \leq \frac{2(f(\boldsymbol{\theta}^0) - f(\boldsymbol{\theta}^*))}{\beta^K \mathcal{T}_T} + \frac{2Z_2 \sum_{t=0}^{T-1} \gamma_t^2}{\beta^K \mathcal{T}_T}. \quad (3.23)$$

Proof. See Appendix A.1.2. □

Unlike conducting convergence analysis in convex problems, the *ergodic convergence* (same as [13, 69]) is adopted as the metric for convergence analysis. Theorem 3.3.2 indicates that, for a randomly selected index $\mathbf{q}$ from $\{0, 1, \dots, T-1\}$ with probability $\{\gamma_{\mathbf{q}}/\mathcal{T}_T\}$, $\mathbb{E}\{\|\bar{\mathbf{g}}_{\boldsymbol{\theta}}^{\mathbf{q}}\|_2^2\}$ is bounded by the RHS of (3.23).

Corollary 3.3.1. *If the learning rate satisfies 1) non-increasing, 2) $\lim_{T \rightarrow \infty} \sum_{t=0}^{T-1} \gamma_t = \infty$, and 3) $\lim_{T \rightarrow \infty} \sum_{t=0}^{T-1} \gamma_t^2 < \infty$, then the RHS of (3.23) converges to 0.*

According to Corollary 3.3.1, by properly scheduling the learning rate, the expected norm of the gradients could converge to 0, i.e., $\lim_{T \rightarrow \infty} \mathbb{E}\{\|\bar{\mathbf{g}}_{\boldsymbol{\theta}}^{\mathbf{q}}\|_2^2\} = 0$. That is, the proposed method is able to converge to critical points. In addition, the lower bound in (3.23) is shown to increase with a larger K . That is, splitting the network into more modules can slow down the convergence. This is consistent with the understanding of the gradient staleness which increases with more split modules, thereby rendering the network learning more difficult. On the other hand, the convergence can also be achieved by setting an appropriate constant learning rate (see the following theorem).

Theorem 3.3.3. *Let Assumptions 3.3.1 and 3.3.2 hold. Set the learning rate to be a*

3.4. EXPERIMENTS

constant:

$$\gamma = \epsilon \sqrt{\frac{f(\boldsymbol{\theta}^0) - f(\boldsymbol{\theta}^*)}{TZ_2}}$$

where ϵ is a scaling factor such that $L\gamma \leq 1$. Let $\boldsymbol{\theta}^*$ be the global minimizer. Then we have

$$\min_{t \in \{0, 1, \dots, T-1\}} \mathbb{E}\{\|\bar{\mathbf{g}}_{\boldsymbol{\theta}}^t\|_2^2\} \leq \frac{(2 + 2\epsilon^2)}{\epsilon\beta^K} \sqrt{\frac{(f(\boldsymbol{\theta}^0) - f(\boldsymbol{\theta}^*))Z_2}{T}}. \quad (3.24)$$

Proof. See Appendix A.1.3. □

As shown in Theorem 3.3.3, for $T \rightarrow \infty$, the RHS of (3.24) converges to 0. This indicates that expected gradient norm could reach 0, i.e., the FDG converging to critical points.

3.4 Experiments

In this section, experiments on CIFAR-10, CIFAR-100 [70], and ImageNet [71] datasets are conducted to compare the generalization and acceleration abilities among different decoupling methods. These experiments show that the proposed FDG performs comparably to or better than the standard BP as well as the state-of-the-art methods.

3.4.1 Comparison of Classification Performance

Implementation Details: The experiments are conducted on the PyTorch platform in a DGX-1 server with Tesla V100 GPUs. Datasets are pre-processed using standard data augmentation (i.e., random cropping, random horizontal flip and normalizing [5]). The SGD optimizer is adopted with an initial learning rate of 0.1 and

3.4. EXPERIMENTS

Table 3.4: The validation errors of the compared methods on CIFAR-10 for $K = 2$. Results with * are rerun using the same training strategy in this chapter. Note that ResNet-18 has more parameters than ResNet-20, ResNet-56 and ResNet-110 as it has more CNN channels [5].

Dataset	Architecture	# params	BP [67]	DDG [13]	DGL [46]	FR [14]	FDG
(a) CIFAR-10	ResNet-20	0.27M	8.75%/7.78%*	-	-	-	7.92%($\beta=1$)/ 7.23% ($\beta=0.2$)
	ResNet-56	0.46M	6.97%/6.19%*	6.89%/6.63%*	6.77%*	6.07%*	6.20%($\beta=1$)/ 5.90% ($\beta=0.5$)
	ResNet-110	1.70M	6.43%/5.79%*	6.59%/6.26%*	6.50%/6.26%*	5.76%*	5.79%($\beta=1$)/ 5.73% ($\beta=0.2$)
	ResNet-18	11.2M	6.48%/4.87%*	5.00%*	5.21%*	4.80%*	4.82%($\beta=1$)/ 4.79% ($\beta=0.8$)
	ResNet-1202	19.4M	7.93%/5.51%*	-	-	-	5.50%($\beta=1$)/ 5.49% ($\beta=0.5$)
	WRN-28-10	36.5M	4.00%/4.01%*	4.05%	4.12%	3.87%	4.13%($\beta=1$)/ 3.85% ($\beta=0.7$)

a momentum of 0.9. For CIFAR datasets, the weight decay is set to 5×10^{-4} and the models are trained using a batch size of 128 for 300 epochs. The learning rate is divided by 10 at 150, 225 and 275 epochs. For ImageNet dataset, a 224×224 crop is randomly sampled and the networks are trained for 90 epochs and decay the learning rate at 30, 60, and 80 epochs. The weight decay is set to 1×10^{-4} . The learning rate $\gamma_t = 0.01$ is used to warm up the training for 3 epochs. The validation errors of all the experiments are reported at the *last epoch* by the median of 3 runs. In particular, the evaluated networks are initialized with *Kaiming Initialization* [72].

The performances of five methods are compared, including those from the BP [67], DDG [13], FR [14], DGL [46] and the proposed FDG. The DNI [11] is not included as its performance has been shown to deteriorate severely with deeper networks [13].

CIFAR-10: The experiments begin by reporting the classification results on the CIFAR-10 dataset, which includes 50000 training and 10000 validation samples of 32×32 color images in 10 classes. In this experiment, the original network is partitioned into two modules ($K = 2$). To ensure fair comparisons, the *rerun* results of the compared methods using the same training strategy are also given.

The Top1 validation errors are reported in Table 3.4. The FR outperforms the DDG consistent with the claim in [14]. It also outperforms the DGL, and even obtains results slightly surpassing the BP baselines. For the proposed method, without the

3.4. EXPERIMENTS

Table 3.5: The validation errors of the compared methods on CIFAR-100 for $K = 2$. Results with * are rerun using the same training strategy in this chapter.

Dataset	Architecture	# params	BP [67]	DDG [13]	DGL [46]	FR [14]	FDG
(b) CIFAR-100	ResNet-56	0.46M	30.21%/27.68%*	29.83%/28.44 %*	29.51%*	28.39%*	27.87%($\beta=1$)/ 27.49% ($\beta=0.4$)
	ResNet-110	1.70M	28.10%/25.82%*	28.61%/27.16%*	26.80%*	26.31%*	25.73%($\beta=1$)/ 25.43% ($\beta=0.5$)
	ResNet-18	11.2M	22.35%*	22.74%*	22.24%*	22.88%*	22.78%($\beta=1$)/ 22.18% ($\beta=0.5$)
	WRN-28-10	36.5M	19.2%/19.6%*	-	-	-	20.28%($\beta=1$)/ 19.08% ($\beta=0.6$)

Table 3.6: The validation errors of the compared methods on ImageNet for $K = 2$. Results with * are rerun using the same training strategy in this chapter.

Architecture	# params	BP [67]		FDG	
		Top1	Top5	Top1	Top5
ResNet-18	11.2M	29.79%*	10.92%*	29.72%($\beta=1$)/ 29.60% ($\beta=0.3$)	10.42% ($\beta=1$)/10.51%($\beta=0.3$)
ResNet-50	25.6M	23.65%*	7.13% *	23.65% ($\beta=1$)/24.08%($\beta=0.5$)	7.23%($\beta=1$)/7.23%($\beta=0.5$)

GS process, the FDG overtakes the DDG and DGL, and achieves comparable results with the BP baselines. With a GS process, networks trained by the FDG are able to generalize better than those trained by the BP and FR.

Additionally, to show that the FDG is able to handle networks with various widths and depths, the decoupled training for the smaller network (ResNet-20), wider network (WRN-28-10 [6]) and extremely deep network (ResNet-1202) are provided. All of these trained networks also generalize better than those trained by the BP.

CIFAR-100: The CIFAR-100 dataset contains the same number of training and validation samples as those in CIFAR-10 but with 100 classes. Again the *rerun* experiments of the compared methods with the same training strategy are given for fairness. The results are reported in Table 3.5. Although overall the FR still overtakes the DDG and DGL, it falls behind the BP. With the GS process, the FDG beats the BP as well as other state-of-the-art methods. In addition, training WRN-28-10 with FDG outperforms the standard BP.

ImageNet: The FDG is also evaluated on the ImageNet 2012 dataset [71] of 1000 classes, which contains 1.28 million training images and 50000 validation images. the

3.4. EXPERIMENTS

Figure 3.3: (a) Performance of ResNet-18 trained on CIFAR-10 for various K with a maximum splittable number $K = 10$. (b) The difference between the GS process and the learning rate scaling (case of ResNet-18, $K=3$). (c)-(e) Impact of β on the performance conducted on ResNet-20.

ResNet-18 as well as ResNet-50 are trained with their Top1 and Top5 error rates reported in Table 3.6. Similar to the CIFAR cases, the FDG also obtains comparable or better results to BP's. This ImageNet experiment shows that the FDG can handle large-scale datasets.

More Split Modules: In this experiment, the performance of ResNet-56 and WRN-28-10 on CIFAR-10 is studied. These networks are split into $K = 3$ and $K = 4$

Table 3.7: Top 1 errors of ResNet-56 and WRN-28-10 on CIFAR-10 for $K=2, 3, 4$.

	BP	DDG	DGL	FR	FDG ($\beta=1/\beta \neq 1$)
ResNet-56 ($K = 2$)	6.19%	6.60%*	6.77%*	6.07%*	6.20%/5.90%($\beta=0.5$)
ResNet-56 ($K = 3$)	6.19%	6.50%*	8.88%*	6.33%*	6.40%/6.08%($\beta=0.2$)
ResNet-56 ($K = 4$)	6.19%	6.61%*	9.65%*	6.48%*	6.83%/6.14%($\beta=0.3$)
WRN-28-10 ($K = 2$)	4.01%	4.05%*	4.12%*	3.87%*	4.13%/3.85%($\beta=0.7$)
WRN-28-10 ($K = 3$)	4.01%	4.12%*	4.91%*	6.16%*	4.19%/4.07%($\beta=0.5$)
WRN-28-10 ($K = 4$)	4.01%	6.61%*	5.64%*	5.39%*	6.50%/4.42%($\beta=0.5$)

3.4. EXPERIMENTS

modules with each module trained in an independent GPU. The results are shown in Table 3.7 where the validation errors with $K = 2, 3, 4$ are listed. It becomes noticeable that more split modules have caused all these methods to reduce accuracy. However, it is also observed that the GS process allows the FDG to maintain the performance comparable to the BP baseline level. This is an empirical proof that the GS process plays an essential role in reducing the gradient staleness, which becomes even more significant as K increases. To further reveal the impact of K on the classification error, the ResNet-18 (with $\beta = 0.7$) is trained on CIFAR-10 with various K , and its performance is depicted in Figure 3.3(a). In particular, $K = 10$ is the maximum splittable number of modules in ResNet-18, with each module being either one layer or one residual block. In general an increase of error rate is observed as K becomes larger, which is reasonable as a large K amplifies the gradient staleness.

To better appreciate the decoupled learning, learning curves of the compared methods w.r.t. wall time (see WRN-28-10 on CIFAR-10 in Figure 3.4) are also provided. It is observed that the fully decoupled methods (DGL and FDG) are much faster than the DDG and the FR. Detailed and quantitative comparisons in terms of acceleration will be given in the following subsection. On the other hand, the proposed FDG tends to converge more smoothly than the DDG and FR, though DGs are involved in all these methods.

From Tables 3.4-3.6, it is observed that although the gradient staleness exists, the FDG ($K = 2$) tends to outperform the compared methods including the traditional BP. Furthermore, the introduced GS process mitigates the gradient staleness while keeping the gradient uncertainty. Therefore, the FDG can provide more promising performance. However, if K keeps increasing, the gradient staleness grows and begins to reduce the network’s generalization ability to a greater extent than the gain

3.4. EXPERIMENTS

Figure 3.4: (a) The left column images are validation error curves of the compared methods for WRN-28-10 on CIFAR-10 (the top/median/bottom images are for $K = 2/3/4$). (b) The right column images are corresponding training error curves (the top/median/bottom images are for $K = 2/3/4$).

3.4. EXPERIMENTS

contributed by its uncertainty (see Figure 3.3(a)).

The Impact of the GS Process: In Tables 3.4-3.6, it has been shown that the GS process with a proper β could enhance the FDG’s generalization ability. Here an additional empirical evaluation is given by setting various values of β . This evaluation is conducted by training the ResNet-20 ($K = 2, 3, 4$) on CIFAR-10 dataset. As shown in Figures 3.3(c)-(e), the performance is not very sensitive to the values of β . Furthermore, networks trained with $0.2 \leq \beta \leq 0.8$ (marked red in Figures 3.3(c)-(e)) generally outperform those with smaller or larger β (i.e., $\beta < 0.2$ or $\beta > 0.8$) including the case without the GS process (i.e., $\beta = 1$). Intuitively, a small β could reduce the network learning capability due to the rapid decrease of gradient strength in the beginning modules, especially for $K > 2$ as shown in (3.17). On the other hand, networks trained with a large β could be more vulnerable to the gradient staleness. The results in Figures 3.3(c)-(e) provide an empirical guideline for choosing a moderate β ranging from 0.2 to 0.8. Within this range, the FDG could consistently outperform its “no-GS” form, which agrees with the results reported in Tables 3.4-3.6. This is another empirical evidence showing the positive impact of the GS process on a network’s ability to generalize. Although the β values range from 0.2 to 0.8, it is recommended to pick values close to 0.5. This is strategically wise especially when only a few trials are permitted as values close to 0.5 (e.g., 0.3-0.7) appear very frequently in the experiments.

On the other hand, to distinguish the GS process from the learning rate scaling (see discussion in Section 3.2.2), the ResNet-18 ($K = 3$) is trained with various values of β for GS process (i.e., $\mathbf{g}_{\theta_i}^{d_{k,t}} \rightarrow \beta^{K-k} \mathbf{g}_{\theta_i}^{d_{k,t}}$), and for learning rate scaling (i.e., $\gamma_t \rightarrow \beta^{K-k} \gamma_t$), respectively. As shown in Figure 3.3(b), in general, the networks trained involving the GS process slightly outperform those trained with the learning

3.4. EXPERIMENTS

rate scaling. This supports the claim made in this chapter of utilizing the GS process.

3.4.2 Comparison of Acceleration Performance

Experiments are conducted by training ResNet-101 on ImageNet (image size of $224 \times 224 \times 3$) with various K values to compare the acceleration abilities among the decoupling techniques. A relatively deep network is chosen as an example because deeper network encourages more balanced workload allocation among different workers. To ensure fair comparisons, the DDG, FR and DGL are re-implemented in the same framework using the identical communication protocol. In particular, a pipeline parallelization version for the sequential DGL [46] is adopted. This is to encourage a fair comparison of speedup among all the methods by allowing each worker to handle one module alone. The training speed is reported by the number of images per second averaged in 50 iterations after the training is stabilized. The batch size is adjusted to maximize the training speed. In addition, the split locations of the network are tuned to allocate the computation workload as even as possible.

Comparisons Among Decoupling Methods: The acceleration results are shown in Table 3.8 with ideal accelerations included for reference. The ideal accelerations for DDG and FR are calculated based on the empirical evidence [73] suggesting that the backward pass takes about 67% of the whole computation. The ideal acceleration for DGL is similar to that of FDG as the auxiliary network adopted needs negligible computation time. For $K = 2, 3, 4$, the DDG and FR obtain speedups of $1.32\times$, $1.57\times$, $1.68\times$ and $1.19\times$, $1.29\times$, $1.45\times$, respectively. These results are consistent with their ideal accelerations marked by $1.50\times$, $1.81\times$, $2.01\times$ and $1.20\times$, $1.51\times$, $1.72\times$. As shown in Table 3.8, the fully decoupled methods provide higher accelerations than those by the DDG and the FR. This is because the fully decoupled methods have

3.4. EXPERIMENTS

Table 3.8: Speed comparisons of ResNet-101 among the compared methods on ImageNet. The communication ratio (Cm. ratio) is also given to measure the communication cost in the proposed FDG.

ResNet-101 ($K = 2$) on ImageNet				ResNet-101 ($K = 3$) on ImageNet			
	images/s (GPU utilizations)	Id. acc.	Cm. ratio	images/s (GPU utilizations)	Id. acc.	Cm. ratio	
BP	220.5 (98%)	1×	-	220.5 (98%)	1×	-	
DDG	291.8 (83%,88%), 1.32×	≈1.50×	-	346.9 (56%,63%,51%), 1.57×	≈1.81×	-	
FR	262.4 (75%,67%), 1.19×	≈1.20×	-	285.0 (65%,57%,69%), 1.29×	≈1.51×	-	
DGL	401.3 (97%,89%), 1.82×	≈2×	-	576.0 (85%,90%,88%), 2.61×	≈3×	-	
FDG	421.5 (95%,90%), 1.91×	2×	0.185%	594.0 (91%,82%,77%), 2.69×	3×	0.137%	

ResNet-101 ($K = 4$) on ImageNet			
	images/s (GPU utilizations)	Id. acc.	Cm. ratio
BP	220.5 (98%)	1×	-
DDG	369.9 (54%,45%,46%,43%), 1.68×	≈2.01×	-
FR	320.0 (35%,67%,58%,61%), 1.45×	≈1.72×	-
DGL	748.8 (82%,84%,89%,85%), 3.39×	≈4×	-
FDG	732.6 (92%,79%,82%,86%), 3.32×	4×	0.450%

Note: To appreciate the speedup, the GPU utilization and the speedup ratio are included. “421.5 (95%,90%), 1.91×” indicates 421.5 images/s with utilizations of 95% and 90% in two GPUs, and a 1.91× acceleration compared with the BP.

Cm. ratio: The time percentage used by communication.

Id. acc.: Ideal acceleration.

much higher ideal accelerations, and tend to use the GPU more efficiently (higher GPU utilizations). Overall, the FDG and the DGL achieve comparable speedups but the FDG delivers more promising classification results especially for $K > 2$ (see Table 3.7). For $K = 2$, the proposed FDG achieves a **1.91×** speedup using 2 GPUs. This is very close to the linear speedup shown in Table 3.3, empirically demonstrating the realization of addressing all the lockings with an ideal speedup of 2×. For $K = 3, 4$, the FDG obtains the speedups of **2.69×** and **3.32×** respectively, with the ideal speedups being 3× and 4×. In general, the FDG accelerates the learning by an average speedup (defined by speedup/ K) of 83%-96% for an additional GPU, which is close to the ideal 100%. This is consistent with the claim of a fully decoupled learning that corresponds to a linear speedup. For demonstration purpose, it is also observed that the speedup results are comparable with those from the state-of-the-art data parallelization (around 90%) [7].

3.4. EXPERIMENTS

Communication Cost: In the proposed FDG, the communication across different workers is essential, yet relatively inexpensive. As reported in Table 3.8, the communication only takes 0.45% of the training expense. For instance, in the experiments, the FDG ($K = 4$) spends around 175 milliseconds to process a batch of 128 images but only about 0.8 milliseconds are allocated to passing the activations and gradients across different workers. Note that unlike in the data parallelization [7], the communication time here is measured without any overlap with the computation.

It is noted that the average speedup slightly decreases for more GPUs. According to Table 3.8, the FDG gives a 96% (1.91/2) average speed gain for a 2nd GPU, but the average speed gain drops to 90% (2.69/3) and 83% (3.32/4) for the 3rd and 4th GPUs. Since it has been known that the communication is not the bottleneck, the main cause of this slow-down is likely the uneven distribution of computation in each worker. This can be indicated by the GPU utilizations reported in Table 3.8. For $K = 3, 4$, the GPU utilizations are less balanced among different workers. This suggests that some GPUs are working at nearly full capacity ($>90\%$) while others ($<80\%$) might be idle for some time. Such imbalanced workload is inevitable in the decoupled learning without a custom design to evenly distribute the computation into each worker. The imbalance becomes more observable as K increases. Experiments with $K \geq 5$ were not included as they would introduce more imbalance such that the acceleration could be slowed down when more GPUs are utilized.

In summary, the proposed FDG has been shown to train various networks in a fully decoupled manner with no or very limited performance decrease. The FDG can also handle large-scale decoupled learning, such as training networks on ImageNet. Through the module-wise model parallelization, the proposed method achieves impressive speed gains (83%-96% average speedup), which are close to the ideal accel-

eration. It is noticed that for large K , the acceleration becomes less efficient due to the uneven distribution of computation in each worker. This inefficiency is currently unresolved in decoupled learning and will be mitigated in future work.

3.5 Summary

In this chapter, a fully decoupled method using delayed gradients (FDG) has been proposed to break the forward, backward, and update lockings for neural network learning. The breaking of these lockings leads to a module-wise parallelization, which enables the network to achieve up to a linear speedup in the ideal case. To enhance the FDG, the gradient shrinking process is introduced and has been empirically shown to improve the network’s ability to generalize, especially when the network is split into more modules. Theoretical analysis has shown that the FDG can converge to critical points of the non-convex optimization problem. The proposed method is validated by training CNNs on CIFAR-10, CIFAR-100 and ImageNet. In these classification tasks, the FDG outperforms the state-of-the-arts and even overtakes the standard BP while achieving a significant acceleration (e.g., $1.91\times$ with 2 GPUs and $2.69\times$ with 3 GPUs). The FDG also succeeds in training networks with various structures, e.g., extremely deep ResNet-1202 and wide WRN-28-10.

Chapter 4

Accumulated Decoupled Learning with Gradient Staleness Mitigation

In the area of decoupled learning, the DG-based methods in general outperforms the LEL-based methods in terms of generalization ability (e.g., see Section 3.4). However, they produce gradient staleness that affects the network’s generalization and convergence during learning. In particular, the staleness grows as the PS (i.e., the number of modules partitioned) increases. This is because the lower modules trained with a large PS need to adopt much older gradients than those trained with a small PS. For demonstration, in the FDG algorithm proposed in Chapter 3, staleness measurement has been prototyped by $2(K - k)$ (e.g., see (3.15) and Figure 3.1), i.e., the steps of gradient delays experienced by the k^{th} module with a PS of K .

To the best of our knowledge, there has been little systematic solutions to mitigate such staleness. The effect of the gradient staleness is either ignored or deliberately avoided in the existing literature of decoupled learning. As a result, only small PS values have been tested in the current methods. For instance, the DDG [13], the

Table 4.1: Characteristics of decoupled leaning methods and the proposed ADL.

Methods	Traditional DG-based methods	LEL-based methods	ADL (ours)
Generalization v.s BP	comparable	worse	comparable
Partition size	small	large	large

Table 4.2: Table of Notations.

Notations	Description
$\hat{\mathbf{g}}_{\theta_l}^{U_{s+j}}$	Processing batch U_{s+j} with the network updated for s times.
LoS	Level of staleness.
$d_{k,j}$	The j^{th} LoS in module k during the gradient accumulation.
$\bar{d}_k$	Averaged LoS.
$\theta_l^{U_s}$	Parameter vector at layer l with the network updated s times.

FR [14], and the recently proposed diversely stale parameters (DSP) in [74] only involve experiments with $\text{PS} \leq 4$.

In summary, the existing DG-based methods can give better generalization results than LEL-methods but they are constrained by a limited PS due to the staleness issue. Dwelling on the DG-based methods, the motivation is to scale the learning to a large PS with comparable generalization performance (see Table 4.1). To this end, the key is to reduce the gradient staleness. In this chapter, the accumulated decoupled learning (ADL), a locking-free decoupled learning with an emphasis on mitigating the gradient staleness, is proposed. The ADL incorporates a module-wise gradient accumulation (mGA) technique in the partitioned modules to mitigate the gradient staleness.

The rest of this chapter is presented as follows. Section 4.1 gives the preliminaries that prepare this chapter for the development of ADL. In Section 4.2, the algorithmic details of the proposed ADL are elaborated. The convergence analysis is given in Section 4.3. Experimental validation is presented in Section 4.4. The chapter is then summarized in Section 4.5.

4.1 Preliminaries

In this section, the related background knowledge is given. The notations here are consistent with those used in Chapter 3. Particularly, the basic definitions of BP and its lockings are not repeated as they have been given in Section 3.1. Instead, the revisit is started with the generic gradient accumulation (gGA) adopted in BP, and then followed by an explanation of the gradient staleness. For convenience, in addition to Table 3.2, frequently used notations in this chapter are presented in Table 4.2.

4.1.1 Learning with Generic Gradient Accumulation

The gGA has frequently been adopted to increase the mini-batch size for training networks on devices with a relatively limited memory setting. The gradients obtained based on several mini-batches are accumulated before they are finally applied to update the network.

To describe the training development involving the gGA technique, an *update index* s and a *wrapped batch index* U_s w.r.t. the original batch index t are introduced. We use the update index s to indicate the s^{th} parameter update of the network. It is connected to U_s in the way of $U_s = Ms$ given M gGA steps. The gGA renders the network parameters unchanged for M steps, i.e., $\theta_l^{U_s} = \theta_l^{U_s+1} = \dots = \theta_l^{U_s+M-1}$. Inversely, the update index can be identified from a batch index t by

$$s = \lfloor t/M \rfloor \tag{4.1}$$

where $\lfloor x \rfloor = \max\{n \in \mathbb{Z} | n \leq x\}$ is the *floor* operator. That is, when the network is processing the t^{th} mini-batch of data, the network has been updated for s times

4.1. PRELIMINARIES

based on (4.1).

Assume that the gradients w.r.t. batch indexes $t = U_s, U_s + 1, \dots, U_s + M - 1$ are accumulated. These gradients are obtained through

$$\mathbf{g}_{\boldsymbol{\theta}_l}^t = \frac{\partial f_{\mathbf{x}_t}(\boldsymbol{\theta}^t)}{\partial \boldsymbol{\theta}_l^{U_s}} \quad (4.2)$$

where parameter $\boldsymbol{\theta}_l^{U_s} = \boldsymbol{\theta}_l^{U_{\lfloor t/M \rfloor}}$ is adopted compared with $\boldsymbol{\theta}_l^t$ in (3.5) to emphasize that the gradients w.r.t. these data batches are obtained based on the same parameter. Using the gGA technique, the weights are updated as follows:

$$\boldsymbol{\theta}_l^{U_{s+1}} = \boldsymbol{\theta}_l^{U_s} - \gamma_s \frac{1}{M} \sum_{j=0}^{M-1} \mathbf{g}_{\boldsymbol{\theta}_l}^{U_s+j}. \quad (4.3)$$

4.1.2 Gradient Staleness

Normally the network is updated with gradients obtained w.r.t. the current parameters. However, there are certain scenarios where the network has to update its parameters with gradients calculated based on "older" parameters. This introduces the gradient staleness or the stale gradient effect, as the gradients are not up-to-date, and are therefore less accurate.

We define the *level of staleness* (LoS) as the update index difference between the current parameter and the parameter used to calculate the stale gradient. That is, assume that a network is updated through

$$\boldsymbol{\theta}_l^{t+1} = \boldsymbol{\theta}_l^t - \gamma_t \mathbf{g}_{\boldsymbol{\theta}_l}^{t-d}, \quad l = 1, \dots, \mathcal{L} \quad (4.4)$$

where $\mathbf{g}_{\boldsymbol{\theta}_l}^{t-d} = \partial f_{\mathbf{x}_{t-d}}(\boldsymbol{\theta}^{t-d}) / \partial \boldsymbol{\theta}_l^{t-d}$. For a gGA step of M , the LoS can be calculated

4.2. AN ACCUMULATED DECOUPLED LEARNING

Figure 4.1: Training a 3-layer network by ADL with $K = 3$ and mGA steps of (a) $M = 2$ and (b) $M = 4$, suggesting that each module is updated every 2 and 4 iterations respectively. Note that there is a batch index difference of $2(K - k)$ between the forward and backward pass. A larger M reduces the LoS according to Eq. (4.5). For instance, at $t = 5$ module 1 (gray) has LoS of 2 with $M = 2$, which is reduced to 1 with $M = 4$.

through

$$\text{LoS} = \lfloor t/M \rfloor - \lfloor (t-d)/M \rfloor \quad (4.5)$$

indicating that the current parameter is $\theta_l^{U_{\lfloor t/M \rfloor}}$ while the parameter used to calculate gradient $g_{\theta_l}^{t-d}$ is $\theta_l^{U_{\lfloor (t-d)/M \rfloor}}$.

4.2 An Accumulated Decoupled Learning

In this section, the algorithmic details of the proposed ADL are presented, including an asynchronous pipelining procedure to achieve model parallelism, and an mGA technique to mitigate the gradient staleness. In particular, the staleness in each module is explicitly shown to decrease.

Prior to our development, like in the FDG in Chapter 3, the network is first

4.2. AN ACCUMULATED DECOUPLED LEARNING

partitioned depth-wise into K modules with a stack of layers in each module. That is, the set of layer indices $\{1, \dots, \mathcal{L}\}$ is partitioned into $\{q(1), q(2), \dots, q(K)\}$ where $q(k) = \{m_k, m_k + 1, \dots, m_{k+1} - 1\}$ denotes the layer indices in module k . This leads to certain notation changes as follows:

$$\begin{aligned}\boldsymbol{\theta}^t &= [(\boldsymbol{\theta}_{q(1)}^t)^T, \dots, (\boldsymbol{\theta}_{q(K)}^t)^T]^T, & \boldsymbol{\theta}_{q(k)}^t &= [(\boldsymbol{\theta}_{m_k}^t)^T, \dots, (\boldsymbol{\theta}_{m_{k+1}-1}^t)^T]^T \\ \boldsymbol{g}_{\boldsymbol{\theta}}^t &= [(\boldsymbol{g}_{\boldsymbol{\theta}_{q(1)}}^t)^T, \dots, (\boldsymbol{g}_{\boldsymbol{\theta}_{q(K)}}^t)^T]^T, & \boldsymbol{g}_{\boldsymbol{\theta}_{q(k)}}^t &= [(\boldsymbol{g}_{\boldsymbol{\theta}_{m_k}}^t)^T, \dots, (\boldsymbol{g}_{\boldsymbol{\theta}_{m_{k+1}-1}}^t)^T]^T \\ \bar{\boldsymbol{g}}_{\boldsymbol{\theta}}^t &= [(\bar{\boldsymbol{g}}_{\boldsymbol{\theta}_{q(1)}}^t)^T, \dots, (\bar{\boldsymbol{g}}_{\boldsymbol{\theta}_{q(K)}}^t)^T]^T, & \bar{\boldsymbol{g}}_{\boldsymbol{\theta}_{q(k)}}^t &= [(\bar{\boldsymbol{g}}_{\boldsymbol{\theta}_{m_k}}^t)^T, \dots, (\bar{\boldsymbol{g}}_{\boldsymbol{\theta}_{m_{k+1}-1}}^t)^T]^T.\end{aligned}$$

We depict the proposed ADL with an example of training a 3-layer network with a PS of $K = 3$ in Figure 4.1(a) ($M = 2$) and 4.1(b) ($M = 4$) respectively. As illustrated in the figures, at every iteration, each module runs a forward and a backward pass. The forward pass is executed with a module input that comes from the output of the lower module at the previous instance. The backward pass calculates the gradients by resuming a local BP using gradients inherited from the upper module based on the "older" data batches. Note that all the partitioned modules can run in parallel due to asynchronism as the modules process data from different batches. Next, each module accumulates gradients for M steps before the gradients are applied to update the module weights.

4.2.1 Asynchronous Learning without BP Lockings

Assume that the weights of module k ($k = 1, \dots, K$) are at update index s with $\boldsymbol{\theta}_{q(k)}^{U_s}$. We detail the learning procedures in module k to conduct update $s + 1$ as follows.

Forward Pass

Module k conducts the forward passes using data batches with indexes $U_s, U_s + 1, \dots, U_s + M - 1$. Let $j = 0, 1, \dots, M - 1$. In detail, the module input $\mathbf{z}_{m_k-1}^{U_s+j}$ received from module¹ $k - 1$ is fed into module k to generate activations in each layer, which are obtained w.r.t. the same parameter $\boldsymbol{\theta}_{q(k)}^{U_s}$. Next, the activation $\mathbf{z}_{m_{k+1}-1}^{U_s+j}$ at the end of this module is obtained and sent to module $k + 1$ (if any).

Backward Pass

During the backward pass, module k resumes BP locally using the gradient $\mathbf{g}_{\mathbf{z}_{m_{k+1}-1}}^{U_s+j-2(K-k)}$ received from module² $k + 1$. Note that the superscript $U_s + j - 2(K - k)$ indicates that there are $2(K - k)$ steps of batch index delay w.r.t. the forward pass (see Figure 4.1 for illustration). Accordingly, the gradient at each layer ($m_k \leq l \leq m_{k+1} - 1$) within this module is calculated as follows:

$$\hat{\mathbf{g}}_{\boldsymbol{\theta}_l}^{U_s+j} = \frac{\partial \mathbf{z}_{m_{k+1}-1}^{U_s+j-2(K-k)}}{\partial \boldsymbol{\theta}_l^{U_{\lfloor (U_s+j-2(K-k))/M \rfloor}}} \mathbf{g}_{\mathbf{z}_{m_{k+1}-1}}^{U_s+j-2(K-k)}. \quad (4.6)$$

Note that (4.6) is obtained w.r.t. $\boldsymbol{\theta}_l^{U_{\lfloor (U_s+j-2(K-k))/M \rfloor}}$ with update index $\lfloor (U_s + j - 2(K - k))/M \rfloor$ instead of s . This is because the gradient is calculated based on the "older" data batches, which can tell their corresponding update indexes from (4.1). At the end of the local BP, gradient $\mathbf{g}_{\mathbf{z}_{m_k-1}}^{U_s+j-2(K-k)}$ w.r.t. the module input $\mathbf{z}_{m_k-1}^{U_s+j-2(K-k)}$ is generated, which is then sent to module $k - 1$ (if any).

¹For $k = 1$ the module input is the training data.

²For $k = K$ the gradient is generated by the loss function.

4.2.2 Update with Module-wise Gradient Accumulation

After obtaining the gradients using (4.6), the module is not updated immediately. Instead, the mGA is applied by accumulating these gradients for M steps before the corresponding module is updated as follows:

$$\boldsymbol{\theta}_l^{U_{s+1}} = \boldsymbol{\theta}_l^{U_s} - \gamma_s(1/M) \sum_{j=0}^{M-1} \hat{\mathbf{g}}_{\boldsymbol{\theta}_l}^{U_s+j}. \quad (4.7)$$

Note that all the modules must consistently accumulate gradients calculated w.r.t. the same group of data batches, though the updates would happen asynchronously. For instance, with an mGA step of 2, if module 1 decides to accumulate batch 10 and 11, the other modules must accumulate these two batches, instead of other possible combinations such as 9 and 10, or 10 and 11. This module-wise nursing step differentiates the mGA from the gGA in the synchronized learning. We summarize the proposed ADL in Algorithm 4.1.

Note that the above ADL is a locking-free decoupled technique. Firstly, the global BP is cast into local BPs in each module running in parallel, which removes the backward locking. Secondly, the split modules adopt training data from different batches so that the forward passes can be executed without waiting for the data from the lower layers. This tackles the forward locking. Finally, each module is updated immediately without waiting for other modules to complete their forward passes, hence addressing the update locking.

4.2.3 Impact of Module-wise Gradient Accumulation

Indicated by (4.6), the gradients are obtained based on $\boldsymbol{\theta}^{U_{\lfloor (U_s+j-2(K-k))/M \rfloor}}$ while the parameter state is $\boldsymbol{\theta}^{U_s}$. Therefore, according to (4.5), the LoS for module k is shown

4.2. AN ACCUMULATED DECOUPLED LEARNING

Algorithm 4.1 The proposed ADL

Partition the network into K modules, and set mGA step of M ;

for *each iteration* **do**

for $k \leftarrow 1$ **to** K (**Parallel**) **do**

Forward pass: generate the activations with module input (e.g., $\mathbf{z}_{m_k-1}^{U_s+j}$), and send the module output (e.g., $\mathbf{z}_{m_{k+1}-1}^{U_s+j}$) to module $k+1$ (if any);

Backward pass: using gradient (e.g., $\mathbf{g}_{z_{m_{k+1}-1}}^{U_s+j-2(K-k)}$) received from module $k+1$ to calculate the gradients in each layer following (4.6), and send the gradient w.r.t. the module input (e.g., $\mathbf{g}_{z_{m_k-1}}^{U_s+j-2(K-k)}$) to module $k-1$ (if any);

Update: **if** module k accumulates gradients from batches $U_s, U_s+1, \dots, U_s+M-1$ **then**

 Update the module using (4.7);

end

end

end

as follows ($j = 0, 1, \dots, M-1$):

$$d_{k,j} = s - \lfloor (U_s + j - 2(K-k))/M \rfloor. \quad (4.8)$$

For instance, as shown in Figure 4.1(b), with $M=4$ module 2 updates its parameters using gradients with staleness $d_{k,j}$ ($1 \leq k \leq K, 0 \leq j < M$) of $d_{2,0} = 1$, $d_{2,1} = 1$, $d_{2,2} = 0$, and $d_{2,3} = 0$. (4.8) also indicates the staleness range:

$$0 \leq d_{k,j} \leq 2(K-k) \quad (4.9)$$

with the minimum $d_{k,j}$ reached for $j-2(K-k) > 0$, and the maximum $d_{k,j} = 2(K-k)$ obtained at $M=1$ indicating no mGA involved. For convenience, to evaluate the staleness in module k , the *averaged LoS* is adopted as follows:

$$\bar{d}_k = (1/M) \sum_{j=0}^{M-1} d_{k,j}. \quad (4.10)$$

Figure 4.2: The averaged LoS w.r.t. mGA step of M .

As an example, Figure 4.2 shows the averaged LoS in module 1 and 6 w.r.t. the accumulation step M with $K = 8$, where the gradient staleness is shown to reduce with increasing M .

Larger M leads to lower gradient staleness, but does not necessarily guarantee improvement in network generalization. This is because larger M also indicates that the overall mini-batch size is large, which could weaken the network’s ability to generalize [75]. As a result, M is an additional hyperparameter that handles the delicate balance between the gradient staleness and the generalization. In fact, large M is not needed for helping the optimization because small M has a rather significant impact on the staleness reduction. As shown in Figure 4.2, with $M = 4$ the averaged LoS is already reduced by 75% (e.g., from 16 to 4).

4.2. AN ACCUMULATED DECOUPLED LEARNING

In addition, using (4.8), (4.6) can be unpacked:

$$\begin{aligned}
\hat{\mathbf{g}}_{\boldsymbol{\theta}_l^{U_s}}^{U_s+j} &= \frac{\partial \mathbf{z}_{m_{k+1}-1}^{U_s+j-2(K-k)}}{\partial \boldsymbol{\theta}_l^{U_{s-d_{k,j}}}} \frac{\partial f_{\mathbf{x}_{U_s+j-2(K-k)}}(\boldsymbol{\theta}^{U_{s-d_{k,j}}})}{\partial \mathbf{z}_{m_{k+1}-1}^{U_s+j-2(K-k)}} \\
&= \frac{\partial f_{\mathbf{x}_{U_s+j-2(K-k)}}(\boldsymbol{\theta}^{U_{s-d_{k,j}}})}{\partial \boldsymbol{\theta}_l^{U_{s-d_{k,j}}}} \\
&= \mathbf{g}_{\boldsymbol{\theta}_l}^{U_s+j-2(K-k)}
\end{aligned} \tag{4.11}$$

which rewrites (4.7) as

$$\boldsymbol{\theta}_l^{U_{s+1}} = \boldsymbol{\theta}_l^{U_s} - \gamma_s(1/M) \sum_{j=0}^{M-1} \mathbf{g}_{\boldsymbol{\theta}_l}^{U_s+j-2(K-k)}. \tag{4.12}$$

That is, the proposed ADL accumulates gradients that are $2(K-k)$ steps “older”, while each of these accumulated gradients admits an LoS of $d_{k,j}$ as shown in (4.8).

4.2.4 Other Methods Using Delayed Gradients

Asynchronous stochastic gradient descent (ASGD) based methods [53, 54] also adopt DGs to facilitate asynchronous distributed learning. They belong to the area of data parallelism since each worker handles the calculation of gradients based on the whole network. By involving DGs, likewise the ASGD-based methods suffer from the gradient staleness. Pipeline distributed methods [76, 77] adopt a similar way of splitting a network depth-wise to that of decoupled learning for acceleration, which also involve DGs and are vulnerable to the gradient staleness.

4.3 Convergence Analysis

In this section, convergence analysis of the ADL is presented. The analysis shows that the ADL can converge to critical points based on the following assumptions.

Assumption 4.3.1. *Lipschitz continuity of gradients for loss functions $f(\boldsymbol{\theta})$, which means $\exists L \in \mathbb{R}^+$ such that:*

$$\|\bar{\mathbf{g}}_{\boldsymbol{\theta}_l}^{U_\alpha} - \bar{\mathbf{g}}_{\boldsymbol{\theta}_l}^{U_\beta}\|_2 \leq L \|\boldsymbol{\theta}_l^{U_\alpha} - \boldsymbol{\theta}_l^{U_\beta}\|_2 \quad (4.13)$$

where $\|\cdot\|_2$ is an l_2 -norm operator. This also leads to

$$\|\bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^{U_\alpha} - \bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^{U_\beta}\|_2 \leq L \|\boldsymbol{\theta}_{q(k)}^{U_\alpha} - \boldsymbol{\theta}_{q(k)}^{U_\beta}\|_2, \quad \|\bar{\mathbf{g}}_{\boldsymbol{\theta}}^{U_\alpha} - \bar{\mathbf{g}}_{\boldsymbol{\theta}}^{U_\beta}\|_2 \leq L \|\boldsymbol{\theta}^{U_\alpha} - \boldsymbol{\theta}^{U_\beta}\|_2.$$

Assumption 4.3.2. *Bounded variance of the stochastic gradient, which means that $\forall s, \exists A > 0$ such that:*

$$\|\mathbf{g}_{\boldsymbol{\theta}_l}^{U_s}\|_2^2 \leq A, \text{ which leads to } \|\mathbf{g}_{\boldsymbol{\theta}_{q(k)}}^{U_s}\|_2^2 \leq A, \quad \|\mathbf{g}_{\boldsymbol{\theta}}^{U_s}\|_2^2 \leq A. \quad (4.14)$$

Assumptions 4.3.1 and 4.3.2 are commonly made for convergence analysis in neural networks (see [13, 69]). In particular, these assumptions do not assume convexity of function f .

Theorem 4.3.1. *Let Assumptions 4.3.1 and 4.3.2 hold. Suppose that the learning rate is non-increasing and $L\gamma_s \leq 1$. The proposed ADL has the following lower bound:*

$$\mathbb{E}_{\mathbf{x}}\{f(\boldsymbol{\theta}^{U_{s+1}})\} - f(\boldsymbol{\theta}^{U_s}) \leq -\frac{\gamma_s}{2} \|\bar{\mathbf{g}}_{\boldsymbol{\theta}}^{U_s}\|_2^2 + \frac{\gamma_s^2 AL(1 + (1/M)\sum_{k=1}^K \bar{d}_k)}{M}. \quad (4.15)$$

Proof. See Appendix A.2.1. □

4.3. CONVERGENCE ANALYSIS

Theorem 4.3.1 gives an important indication for convergence. If the RHS of (4.15) is negative, i.e.,

$$\gamma_s < \min \left\{ 1/L, \frac{M \|\bar{\mathbf{g}}_{\boldsymbol{\theta}}^{U_s}\|_2^2}{2AL(1 + (1/M)\sum_{k=1}^K \bar{d}_k)} \right\},$$

the expected loss $\mathbb{E}_{\mathbf{x}}\{f(\boldsymbol{\theta}^{U_{s+1}})\}$ decreases. We further give the convergence evidence in the following theorems.

Theorem 4.3.2. *Suppose Assumptions 4.3.1 and 4.3.2 hold, and the learning rate is non-increasing as well as satisfies $L\gamma_s \leq 1$. Let $\boldsymbol{\theta}^*$ be the global minimizer and $\mathbb{T}_S = \sum_{s=0}^{S-1} \gamma_s$ where S indicates the network will be updated S times. Then*

$$\frac{1}{\mathbb{T}_S} \sum_{s=0}^{S-1} \gamma_s \mathbb{E}\{\|\bar{\mathbf{g}}_{\boldsymbol{\theta}}^{U_s}\|_2^2\} \leq \frac{2AL(1 + (1/M)\sum_{k=1}^K \bar{d}_k) \sum_{s=0}^{S-1} \gamma_s^2}{M\mathbb{T}_S}. \quad (4.16)$$

Proof. See Appendix A.2.2. □

The lower bound in Theorem 4.3.2 indicates that, for a randomly selected $\mathbf{q}$ from $\{0, 1, \dots, S-1\}$ with probability $\{\gamma_{\mathbf{q}}/\mathbb{T}_S\}$, $\mathbb{E}\{\|\bar{\mathbf{g}}_{\boldsymbol{\theta}}^{U_s}\|_2^2\}$ is bounded by the RHS of (4.16). More importantly, a larger M leads to a smaller lower bound in (4.16) because the $\bar{d}_k$ decreases, and thus benefits the convergence. Another observation is that larger PS hinders the convergence as $\sum_{k=1}^K \bar{d}_k$ increases. These observations are consistent with the understanding that the mGA helps the optimization by mitigating staleness, and partitioning the network into more modules is harmful.

Corollary 4.3.1. *If γ_s further satisfies $\lim_{S \rightarrow \infty} \mathbb{T}_S = \infty$ and $\lim_{S \rightarrow \infty} \sum_{s=0}^{S-1} \gamma_s^2 < \infty$, the RHS of (4.16) converges to 0.*

According to Corollary 4.3.1, by properly scheduling the learning rate, the lower bound for the expected gradient would converge to 0, i.e., $\lim_{S \rightarrow \infty} \mathbb{E}\{\|\bar{\mathbf{g}}_{\boldsymbol{\theta}}^{U_s}\|_2^2\} =$

4.4. EXPERIMENTS

0. That is, the proposed ADL can converge to critical points. Alternatively, the convergence can be revealed by setting a constant learning rate as indicated in the following theorem.

Theorem 4.3.3. *Let Assumptions 4.3.1 and 4.3.2 hold. Suppose the learning rate is set as a constant:*

$$\gamma = \epsilon \sqrt{\frac{M(f(\boldsymbol{\theta}^0) - f(\boldsymbol{\theta}^*))}{SAL(1 + \sum_{k=1}^K \bar{d}_k)}}$$

where ϵ is a scaling factor such that $L\gamma \leq 1$. Let $\boldsymbol{\theta}^*$ be the global minimizer. Then we have

$$\min_{s \in \{0, 1, \dots, S-1\}} \mathbb{E}\{\|\bar{\mathbf{g}}_{\boldsymbol{\theta}}^{U_s}\|_2^2\} \leq \frac{(2 + 2\epsilon^2)}{\epsilon} \sqrt{AL(f(\boldsymbol{\theta}^0) - f(\boldsymbol{\theta}^*)) \left(1 + (1/M) \sum_{k=1}^K \bar{d}_k\right) / (MS)}, \quad (4.17)$$

where the lower bound converges to 0 when $S \rightarrow \infty$.

Proof. See Appendix A.2.3. □

In summary, although the ADL attains model parallelism introducing asynchronism, the proposed ADL can converge to critical points. The analysis has also revealed how the convergence can be affected by the mGA and the PS.

4.4 Experiments

For validation, classification tasks on the CIFAR-10 [70] and ImageNet 2012 [71] datasets are conducted. The main focus is on examining the generalization of networks trained by ADL in order to obtain empirical evidence of gradient staleness

4.4. EXPERIMENTS

Table 4.3: Validation errors (%) for the compared methods training various networks on CIFAR-10. $\otimes$ indicates divergence.

$M \rightarrow$	1	2	3	4	5	6	7	8	9	10	11	12
ResNet-56 ($K = 16$)	$\otimes$	$\otimes$	$\otimes$	6.38	6.90	6.92	6.79	6.49	6.78	6.44	6.65	6.86

mitigation in decoupled learning. Its acceleration performance is then lightly examined with discussion of an imbalance issue affecting the acceleration, which is currently unresolved in the area of decoupled learning. The proposed method is compared with several state-of-the-arts, including DDG [13], FR [14], DGL [46], DSP [74], FDG in Chapter 3 and BP [67]. The comparison happens among methods of decoupled learning, with the ASGD- and pipeline-based methods excluded though they might also involve gradient staleness.

Implementation details: The experiments are performed with PyTorch [78] where the datasets are pre-processed using standard data augmentation (i.e., random cropping, random horizontal flip and normalizing). For all the compared methods (except DSP due to no source code), an identical training strategy to that in the FDG chapter is adopted for fairness. An overall batch size of 128 is used for all methods. Specially, since the mGA accumulates the data batches, for ADL the batch size in each iteration is set to $\lfloor 128/M \rfloor$.

4.4.1 Generalization Performance

Firstly, the impact of mGA is evaluated by training ResNet-56 ($K = 16$) on CIFAR-10 with various M values. As shown in Table 4.3, the ADL diverges with $M \leq 3$ due to strong gradient staleness. Once the learning converges, the ADL performs rather robustly w.r.t. various K . For convenience, $M = 4$ (i.e., reducing 75% staleness as shown in (4.10)) is selected for the ADL in the following experiments, which should

4.4. EXPERIMENTS

stabilize the decoupled learning for $K \leq 16$.

Later on, on CIFAR-10 various ResNet architectures are trained with PS values ranging from $K = 2$ to $K = 16$. The classification results are shown in Table 4.4. In general the ADL delivers results comparable to or better than those from the compared methods, while the other methods of decoupled learning would encounter learning issues as K increases. The DGL experiences a growing loss of performance as the PS increases, e.g., $6.77\% \rightarrow 9.65\% \rightarrow 13.26\%$ for ResNet-56 with $K = 2, 4, 8$ respectively. The DGL severs the gradient flow between adjacent modules. It renders the lower modules under-developed without any feedback from the upper modules, causing inferior learning results to the BP baselines.

The FR gives promising results for small PS values (e.g., $K = 2$) which is consistent with the claim made in [14]. However, its performance shows significant deterioration when training with a very large PS. For instance, training ResNet-56 with $K = 16$ leads to an error rate of 11.51% which is significantly worse than that of 6.07% with $K = 2$. In the case of ResNet-34 ($K = 12$), the FR even fails to converge. The performance drop or divergence is likely resulted from the recomputation unit in FR which adopts inconsistent weights during the forward and backward passes. The DSP gives a moderate performance for training ResNet-98. Other corresponding results are not provided here as no source code is released and it is difficult to reproduce the algorithm based on the manuscript. Nonetheless, a similar behaving pattern of DSP to that of FR would be expected as they share the same recomputation technique.

The DDG and the FDG, on the other hand, are more prone to divergence. As shown in Table 4.4, the trainings of ResNet-56 and ResNet-110 exhibit several divergence cases (mainly happen with large PS, e.g., $K \geq 8$), while the ADL successfully trains these networks with comparable performance to the BP baselines. In partic-

4.4. EXPERIMENTS

Table 4.4: Validation errors (%) for the compared methods training various networks on CIFAR-10. $\otimes$ indicates divergence. The best results are marked by **boldface black** digits while the second best marked by **boldface blue** numbers.

Architecture	BP	DDG	DGL	FR	DSP*	FDG	ADL
RN-56 ($K = 2$)	6.19	6.63	6.77	6.07	-	5.90	5.99
RN-56 ($K = 3$)	6.19	6.50	8.88	6.33	-	6.08	6.09
RN-56 ($K = 4$)	6.19	6.61	9.65	6.48	-	6.14	6.16
RN-56 ($K = 8$)	6.19	$\otimes$	13.26	6.64	-	$\otimes$	6.18
RN-56 ($K = 16$)	6.19	$\otimes$	13.36	11.51	-	$\otimes$	6.38
RN-110 ($K = 2$)	5.79	6.26	6.26	5.76	-	5.73	5.87
RN-110 ($K = 8$)	5.79	$\otimes$	11.96	6.56	-	$\otimes$	5.80
RN-98 ($K = 4$)	6.01	6.19	9.7	6.01	6.59	5.70	5.92
RN-98 ($K = 8$)	6.01	$\otimes$	12.37	6.35	-	$\otimes$	5.93
RN-34 ($K = 12$)	4.50	$\otimes$	9.56	$\otimes$	-	$\otimes$	4.63
RN-164 ($K = 4$)	5.36	5.43	8.77	5.60	5.58	5.74	5.46
RN-164 ($K = 10$)	5.36	5.58	10.70	5.89	-	$\otimes$	5.52

* Only results from DSP’s paper are provided due to no available source code.

ular, when the DDG does converge (e.g., training ResNet-164 with $K = 10$), unlike the FR or DGL with rather sharply decreased performance, it tends to deliver very close results to those of BP (e.g., 5.58% versus 5.36%). Such an observation reveals that the increasing staleness invites instability yet it only affects the generalization to a limited extent. This empirically supports and further justifies the motivation of developing the proposed ADL within the DG-based methods aiming to mitigate gradient staleness thereby avoiding divergence.

The learning curves of ResNet-34 ($K = 12$) and ResNet-56 ($K = 16$) are depicted in Figure 4.3. The DDG and FDG diverge in both cases, i.e., DDG diverged at epoch 48 and 2 for ResNet-34 and ResNet-56 while FDG diverged at epoch 1 for both networks. The FR diverges at epoch 2 for ResNet-34, and experiences a “bumpy” convergence for ResNet-56. The ADL converges smoothly in both cases while achieving equally good validation results to the BP baselines. A clear pattern in these learning curves is also

Figure 4.3: Learning curves of compared methods for training ResNet-34 with $K = 12$ (the 1st column) and ResNet-56 with $K = 16$ (the 2nd column), and the error rates w.r.t. PS (the 3rd column) on ResNet-18. Note that the DSP is excluded as no source code is available.

a

4.4. EXPERIMENTS

Table 4.5: Top1 and Top5 validation errors (%) of BP and ADL on ImageNet.

Architecture	BP	ADL
ResNet-18 ($K = 3$)	29.79/10.92	29.52/10.42
ResNet-18 ($K = 4$)	29.79/10.92	29.64/10.56
ResNet-18 ($K = 8$)	29.79/10.92	29.75/10.55
ResNet-18 ($K = 10$, max.) [†]	29.79 /10.92	29.84/ 10.76
ResNet-50 ($K = 4$)	23.65/ 7.13	23.43 /7.45
SE-ResNet-18 ($K = 8$)	29.09/ 9.89	29.01 /10.14
SE-ResNet-18 ($K = 10$, max.) [†]	29.09/ 9.89	29.07 /10.32

[†] The largest PS with each module being one layer or one residual block.

observed showing that the ADL gives a slightly slower convergence than that from BP at the beginning of network learning (see the zoomed plots in Figure 4.3). This is consistent with the theoretical findings (e.g., Theorem 4.3.2) suggesting the decoupled learning would slow down the convergence especially with a large PS. The study to examine the impact of PS on the error rate is also provided by training ResNet-18 with various K . Figure 4.3(e) shows that the proposed method is relatively robust to the change of K with the staleness mitigated, while the compared methods encounter a more rapid rise of errors as K increases.

The results of ADL trained on ImageNet are also provided in comparison with BP. Several architectures from ResNet and SENet [79] are trained with both Top1 and Top5 error rates reported. As shown in Table 4.5, in general the ADL outperforms its BP counterpart even for a large PS of $K = 10$. The observation of ADL outperforming the BP can be explained as follows. The ADL adopts DGs, which can be treated as real gradients obtained by BP, yet contaminated with noises drawn from an unknown distribution. Although the contaminated gradients could slow down the convergence (e.g., see the zoomed plots in Figure 4.3), they could however improve the network’s generalization with their uncertainties (see [80]). Note that such an ex-

4.4. EXPERIMENTS

Table 4.6: Speedups (over BP) in training ResNet-101 (ImageNet).

	BP	DDG	FR	DGL	DSP	FDG	ADL
K=2	1×	1.32×	1.19×	1.82×	-	1.91	1.92 ×
K=3	1×	1.57×	1.29×	2.61×	-	2.69	2.69 ×
K=4	1×	1.68×	1.45×	3.39 ×	2.70×	3.32	3.33×

planation applies to the FDG case as well. In particular, in ADL the mGA restrains the staleness so it does not go out of bound while keeping its uncertainty, thereby leading to more promising results in certain cases.

4.4.2 Acceleration Performance

Here the acceleration performance of the ADL is shown in comparison with other methods of decoupled learning by training ResNet-101 on ImageNet with various K . The experimental settings (e.g., partition location, GPU model) are identical to that of FDG method. Note that the mGA has little effect on the running speed, so $M = 4$ is selected as usual.

As shown in Table 4.6, by addressing all three lockings of BP, the ADL achieves a remarkable acceleration over BP which is similar to the fully decoupled method FDG proposed in Chapter 3 without its divergence issue (see Table 4.4). The ADL is much faster than the DDG and FR which address only the backward locking. For instance, for $K = 2$ the ADL accelerates the learning with a speedup of **1.92**× while the DDG and FR only obtain **1.32**× and **1.19**× respectively. Although the DSP also addresses all the unlockings, it is lower than the ADL due to the recomputation unit that demands extra computing power. The DGL delivers comparable acceleration results to those of ADL, but it leads to weaker generalization performance (see Table 4.4).

4.4. EXPERIMENTS

Table 4.7: Memory consumption of the compared methods.

	BP	DDG	FR	DGL	FDG/ADL
Mem. (GB)	1.14	1.69	1.46	1.49	1.84

4.4.3 Memory Analysis

Here we conduct a brief comparison regarding the memory consumption among the compared methods. The results are measured based on the average GPU memory in training RN-18 of $K = 4$ with a batch size of 128. As indicated in Table 4.7, the compared methods used more memory than that used by BP. Compared with the FR and GPipe, the DDG and ADL require additional memory. This is because, unlike the trainings in FR, no recomputation is required, which however allow them, especially the ADL, to run much faster (e.g., $3.33\times$ of ADL v.s. $1.45\times$ of FR for 4 GPUs). Reducing the memory usage without affecting the ADL’s generalization and speed is a nontrivial task, which will be our future work. Note that the FDG and ADL would require the same amount of memory as the formulations are similar, and the mGA in ADL does not consume extra memory.

In summary, it has been shown that the proposed ADL can obtain generalization results comparable with that from the compared methods while having significantly better stability than the other DG-based methods. This is achieved because the gradient staleness can be mitigated (e.g., see Figure 4.3), which is consistent to the theoretical analysis (e.g., see Figure 4.2). As a decoupled learning method, the ADL’s acceleration performance is also investigated, which is shown to be among the fastest similar to the FDG proposed in Chapter 3.

4.5 Summary

In this chapter, an accumulated decoupled learning (ADL) algorithm has been proposed to address the inefficiency in BP lockings thereby achieving model parallelism. Particularly, a module-wise gradient accumulation (mGA) technique, which mitigates the gradient staleness that hinders the scaling ability of decoupled learning, has been incorporated. The effect of mitigation has been demonstrated theoretically and empirically. Theoretically, the ADL is shown to converge to critical points, and the convergence analysis has demonstrated how the partition size (PS) and the mGA could affect the lower bound categorizing the convergence. The classification tasks conducted validated the claim made in this chapter as the ADL outperformed the state-of-the-art counterparts especially in the case of a large PS. Finally the running speed was investigated, demonstrating a remarkable acceleration for the ADL as a model parallelism tool.

Chapter 5

Correlation Projection for Analytic Learning

As detailed in Chapter 1 and Section 2.2.1, existing analytic learning methods thrive in shallow (i.e., 2-layer) networks while facing several limitations, such as strong structural constraint [17] and needing invertible activation function [18], in training multilayer networks. This has motivated us to develop new multilayer analytic learning techniques without being constrained by the aforementioned limitations. In this chapter, a correlation projection network (CPNet) for multilayer analytic network learning is proposed. The proposed network contains several 2-layer modules that conduct their own locally supervised learnings. As a new analytic learning technique, the proposed CPNet achieves a multi-exit structure for prediction which is the first of its kind in multilayer analytic learning.

The rest of this chapter is presented as follows. Section 5.1 revisits some background knowledge that helps to develop the main results in the subsequent section. The proposed CPNet is detailed in Section 5.2. The experimental validation can be

5.1. PRELIMINARIES

Table 5.1: Table of Notations.

Notations	Description
$\dagger$	The MP inverse operator.
F_l	Activation function at layer l .
$\mathbf{X}_l$	The input at layer l or activation of the previous layer.
$\mathbf{Y}$	The target matrix.
$\mathbf{Y}_l$	The output at layer l .
$\hat{\mathbf{Y}}_l$	The prediction at layer l . Note the difference from $\mathbf{Y}_l$.
$\tilde{\mathbf{Y}}_l$	The encoded label matrix at layer l .
$\mathbf{W}_l$	The weight matrix at layer l .
$\hat{\mathbf{W}}_l$	The trained weight matrix at layer l .
$\mathbf{P}_l$	The correlation projector at layer l , randomly determined.
$\mathbf{Q}_l$	The label encoder at layer l , randomly determined.
$\mathbf{Q}_l^\dagger$	The label decoder at layer l .

found in Section 5.3. The chapter is then summarized in Section 5.4.

5.1 Preliminaries

In this section, the background knowledge of a multilayer network and its representation in matrix formulation is revisited. In order to avoid confusion, the frequently used notations are listed in Table 5.1. These notations have been adopted throughout this chapter.

5.1.1 Multilayer Network Structure

Given an N -sample dataset $\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^N$ where $\mathbf{x}_i \in \mathbb{R}^{d_o}$ and $\mathbf{y}_i \in \mathbb{R}^{d_y}$ are respectively the data and label vectors in the i^{th} sample. We stack these samples to build the

5.1. PRELIMINARIES

data matrix $\mathbf{X}$ and label matrix $\mathbf{Y}$ as follows:

$$\begin{aligned}\mathbf{X} &= [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N]^T \in \mathbb{R}^{N \times d_0} \\ \mathbf{Y} &= [\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_N]^T \in \mathbb{R}^{N \times d_y}.\end{aligned}$$

To include the bias term, an augmented data matrix can be written as $\mathbf{X}_0 = [\mathbf{1}, \mathbf{X}] \in \mathbb{R}^{N \times (d_0+1)}$. Note that if the label is categorical, it can be converted into a binary vector using one-hot encoding [81].

For an $\mathcal{L}$ -layer MLP, if the last layer is linear, the output of the network can be written as

$$\mathbf{Y}_{\mathcal{L}} = f_{\mathcal{L}-1}(\dots f_2(f_1(\mathbf{X}_0 \mathbf{W}_1) \mathbf{W}_2) \dots) \mathbf{W}_{\mathcal{L}} \quad (5.1)$$

with network output $\mathbf{Y}_{\mathcal{L}} \in \mathbb{R}^{N \times d_y}$, and weight matrices for each layer: $\mathbf{W}_1 \in \mathbb{R}^{d_0 \times d_1}$, $\dots$, $\mathbf{W}_l \in \mathbb{R}^{d_{l-1} \times d_l}$, $\dots$, $\mathbf{W}_{\mathcal{L}} \in \mathbb{R}^{d_{\mathcal{L}-1} \times d_y}$. For convenience, an $\mathcal{L}$ -layer MLP can be presented with a structure of “ $-d_1 - \dots - d_{\mathcal{L}-1} -$ ”. For instance, “-100-200-” indicates a 3-layer network with 200 and 300 units in hidden layers. Here, it is noted that the structure in (5.1) follows that of [17, 61, 82] with inclusion of the bias term only in the input layer. Apart from notational simplicity, this structure without biases in the hidden layers does not appear to have adversary impact towards the results observed. The goal of network learning is to determine $\{\mathbf{W}_1, \dots, \mathbf{W}_{\mathcal{L}}\}$ given the input $\mathbf{X}_0$ and the label $\mathbf{Y}$, such that the output $\mathbf{Y}_{\mathcal{L}}$ approaches the label $\mathbf{Y}$ according to its criterion. That is, one needs to solve the following optimization problem:

$$\min_{\mathbf{W}_1, \dots, \mathbf{W}_{\mathcal{L}}} \text{Loss}(\mathbf{Y}, \mathbf{Y}_{\mathcal{L}}) \quad (5.2)$$

5.2. PROPOSED METHOD

where the *Loss* function measures the similarity between $\mathbf{Y}$ and $\mathbf{Y}_{\mathcal{L}}$. Commonly used loss functions include cross-entropy and mean squared error [83]. The optimization problem (5.2) is highly non-convex, and is usually handled by the BP algorithm through numerous iterative searches in order to reach a suitable local optimum.

5.1.2 Multilayer Analytic Learning

Due to the iterative nature, solving the multilayer network learning problem with BP could incur various problems, such as gradient vanishing/explosion and long training time. For those existing attempts, such as the PIL [17], PILAE [61], KARnet [18], and ANnet [65], they discard the use of gradient BP through analytic network learning. These analytical solutions usually require only one visit of a training dataset, leading to significantly faster learning pace than that of the BP which demands hundreds of training epochs or beyond. For the analytic methods, feeding the label information into hidden layers is an important way to achieve multilayer analytic learning. For instance, the KARnet [18] projects the label into hidden layers through sequential projection with MP inverse operations. The PILAE [61] uses its input as label to supervise the learning in hidden layers. The work in this chapter aligns with these label-feeding learning methods, but differs in terms of network structure and label-feeding technique. This will be detailed in the following section.

5.2 Proposed Method

In this section, the details of the proposed CPNet are elaborated. This includes the training phase, the validation prediction phase, and the testing phase. The training phase shows how CPNet obtains its weights. The validation prediction phase demonstrates how predictions are made during the validation process (e.g., using k -fold cross

5.2. PROPOSED METHOD

validation [84]) for network structure determination. The testing phase evaluates the performance of a trained network. The proposed analytic learning in this network has three key features:

1. A label encoding (LE) process, which encodes the label information into hidden layers with appropriate dimension conversion.
2. A correlation projection (CP) technique to extract features from lower layers, which is then followed by an LS estimation for analytical determination of network weights.
3. A label decoding (LD) process, which allows each module to give an individual prediction when making predictions in the validation phase, achieving multiple exits in a single network.

Prior to our developments, some indexing issues must be highlighted. Unlike the conventional layer indexing of $l = 1, 2, \dots, \mathcal{L}$, layer indexes such as $2m - 1$ and $2m$ will be frequently adopted with module index $m = 1, 2, \dots, M$ where $M = \mathcal{L}/2$ and the depth $\mathcal{L}$ is an even integer. This is due to the module-based (i.e., 2 layers in each module) structure of the proposed CPNet.

5.2.1 Training Phase

The training phase determines the weight matrices in each module. This includes the LE process for target preparation, the CP technique for feature extraction, and the LS estimator for analytical weight determination.

In the training phase (see Figure 5.1(a)), the CPNet is of a module-based structure with each module containing 2 layers (highlighted by green dash boxes). We first introduce the LE process that inserts label information into each module, which

Figure 5.1: The proposed CPNet has a module-based structure with each module containing two layers. **(a)** Training phase: The label information is first encoded into hidden layers through the label encoders. Subsequently, the weights in each module are determined by the CP process in the first layer and the LS solution in the second layer. **(b)** Prediction in the validation phase: Adopting the label decoders, each module can give its own prediction, achieving multiple exits for the same input.

5.2. PROPOSED METHOD

facilitates locally supervised learning. Subsequently, within each module, the first layer employs a CP procedure to extract features from the input while the second layer maps the extracted features analytically onto the encoded label.

Label Encoding

The lack of label information is one of the main causes that hinder multilayer analytic learning. Here it is shown that the label information can be encoded into hidden layers to assist supervised learning in each module.

Our LE process is a direct approach by adopting a linear transformation to accommodate the size difference. Given the label matrix $\mathbf{Y} \in \mathbb{R}^{N \times d_y}$ at the output where d_y is the output size, the LE process (see solid red lines in Figure 5.1(a)) inserts the hidden label into hidden layers indexed by $2m$ as follows:

$$\tilde{\mathbf{Y}}_{2m} = \mathbf{Y} \mathbf{Q}_{2m}, \quad m = 1, \dots, M \quad (5.3)$$

where $\mathbf{Q}_{2m} \in \mathbb{R}^{d_y \times d_{2m}}$ and $\tilde{\mathbf{Y}}_{2m} \in \mathbb{R}^{N \times d_{2m}}$ are respectively referred to as the *label encoder* and the *encoded label matrix* in the $2m^{\text{th}}$ layer. In this chapter, each element of a label encoder is randomly drawn from the normal distribution $\mathcal{N}(0, 1)$. Note that the last module (i.e., module M) is directly linked to the actual label without the LE process, i.e., $\tilde{\mathbf{Y}}_{2M} = \mathbf{Y}$.

This simple but effective encoding trick feeds label information into hidden layers. These encoded label matrices are formed by the combination of the actual labels, with the true label information encrypted through a linear transformation. Such an encoding could be lossless under certain conditions. For instance, the LE process in (5.3) is invertible such that $\mathbf{Y} = \tilde{\mathbf{Y}}_{2m} \mathbf{Q}_{2m}^\dagger = \mathbf{Y} \mathbf{Q}_{2m} \mathbf{Q}_{2m}^\dagger = \mathbf{Y}$ if $\mathbf{Q}_{2m}$ has full row rank. After the label encoding, one could convert a multilayer network into multiple

5.2. PROPOSED METHOD

2-layer modules which are much easier to train.

Correlation Projection and Analytical Weight Determination

After feeding the encoded label, each module has its own label information to guide the learning. Within each module, its weights are determined through the CP process in the first layer and solving the LS problem analytically in the second layer as shown in Figure 5.1(a). The CP process works as follows. In the m^{th} module, the module input is $\mathbf{X}_{2m-2} \in \mathbb{R}^{N \times d_{2m-2}}$. In the first layer, the activation is constructed via:

$$\mathbf{X}_{2m-1} = f_{2m-1}(\mathbf{X}_{2m-2} \mathbf{X}_{2m-2}^T \mathbf{P}_{2m-1}), \quad m = 1, \dots, M \quad (5.4)$$

where the non-zero matrix $\mathbf{P}_{2m-1} \in \mathbb{R}^{N \times d_{2m-1}}$ is a *correlation projector* that projects the correlation matrix $\mathbf{X}_{2m-2} \mathbf{X}_{2m-2}^T \in \mathbb{R}^{N \times N}$ onto $\mathbf{X}_{2m-2} \mathbf{X}_{2m-2}^T \mathbf{P}_{2m-1} \in \mathbb{R}^{N \times d_{2m-1}}$ with a selected hidden layer size d_{2m-1} . We refer to this layer as the *CP layer*. Similar to the selection of label encoders, the elements of the projector $\mathbf{P}_{2m-1}$ are also randomly determined according to the normal distribution $\mathcal{N}(0, 1)$.

The correlation indicated by $\mathbf{X}_{2m-2} \mathbf{X}_{2m-2}^T$ conducts a kernel-like operation, which extracts useful representations from the module input. The correlation projector that follows is able to reshape the correlation matrix according to the determined network structure.

To be memory-friendly, in (5.4) the $\mathbf{X}_{2m-2}^T \mathbf{P}_{2m-1}$ can be calculated first because its size is usually much smaller than that of the correlation matrix $\mathbf{X}_{2m-2} \mathbf{X}_{2m-2}^T$ of size $N \times N$. In such case, (5.4) can be rewritten as a standard form as follows:

$$\mathbf{X}_{2m-1} = f_{2m-1}(\mathbf{X}_{2m-2} \hat{\mathbf{W}}_{2m-1}), \quad m = 1, \dots, M \quad (5.5)$$

5.2. PROPOSED METHOD

where

$$\hat{\mathbf{W}}_{2m-1} = \mathbf{X}_{2m-2}^T \mathbf{P}_{2m-1}, \quad m = 1, \dots, M \quad (5.6)$$

is the actually weight of the CP layer. An important observation is that the weight of this CP layer is determined purely with a matrix multiplication without any feedback or matrix inversion. The randomization involved in determining $\hat{\mathbf{W}}_{2m-1}$ differentiates the CPNet from those random-weighted networks [29, 58]. As shown in (5.6), each row of $\hat{\mathbf{W}}_{2m-1}$ resides in the space spanned by the rows of $\mathbf{X}_{2m-2}^T$, while traditional random weighted networks determine $\hat{\mathbf{W}}_{2m-1}$ with completely randomized elements.

Next, the activation of the CP layer, i.e., $\mathbf{X}_{2m-1}$, is mapped onto the encoded label matrix $\tilde{\mathbf{Y}}_{2m}$. The mapping can be described by the following minimization problem:

$$\min_{\mathbf{W}_{2m}} \|\tilde{\mathbf{Y}}_{2m} - \mathbf{X}_{2m-1} \mathbf{W}_{2m}\|_2, \quad m = 1, \dots, M \quad (5.7)$$

which can be optimally solved by:

$$\hat{\mathbf{W}}_{2m} = \mathbf{X}_{2m-1}^\dagger \tilde{\mathbf{Y}}_{2m}, \quad m = 1, \dots, M. \quad (5.8)$$

Note that (5.8) is an optimal solution to the minimization problem (5.7) based on the MP inverse. It is more general than the left or the right inverse based solutions, i.e., $(\mathbf{X}_{2m-1}^T \mathbf{X}_{2m-1})^{-1} \mathbf{X}_{2m-1}^T \tilde{\mathbf{Y}}_{2m}$ or $\mathbf{X}_{2m-1} (\mathbf{X}_{2m-1} \mathbf{X}_{2m-1}^T)^{-1} \tilde{\mathbf{Y}}_{2m}$, as it automatically takes care of the rank-deficient case when $\text{rank}(\mathbf{X}_{2m-1}) < \min\{d_{2m-1}, N\}$.

After learning the weights in the m^{th} module, the output for this module is written

5.2. PROPOSED METHOD

as

$$\mathbf{Y}_{2m} = \mathbf{X}_{2m-1} \hat{\mathbf{W}}_{2m}, \quad m = 1, \dots, M. \quad (5.9)$$

Subsequently, the module output passes through the activation function, i.e.,

$$\mathbf{X}_{2m} = f_{2m}(\mathbf{Y}_{2m}), \quad m = 1, \dots, M \quad (5.10)$$

to be fed to the successive module as its module input. More specifically, the input for the next module is determined through

$$\mathbf{X}_{2m} = f_{2m}(f_{2m-1}(\dots f_2(f_1(\mathbf{X}_0 \hat{\mathbf{W}}_1) \hat{\mathbf{W}}_2) \dots) \hat{\mathbf{W}}_{2m}), \quad m = 1, \dots, M. \quad (5.11)$$

In summary, module m learns its parameters (i.e., $\hat{\mathbf{W}}_{2m-1}$ and $\hat{\mathbf{W}}_{2m}$) using the input $\mathbf{X}_{2m-2}$ and the label $\tilde{\mathbf{Y}}_{2m}$. We can also view the module-wise learning from a transfer learning perspective. As shown in (5.11), the input for each module can be treated as the feature extracted from a pretrained network that consists of all the learned lower layers. The training phase of the proposed CPNet is described in Algorithm 5.1.

The proposed CPNet is computationally efficient due to its particular structure. Except for the MP inverse computation in (5.8), all the other operations (e.g., matrix multiplications) are highly parallelizable. Therefore, the CPNet could benefit tremendously from modern parallel-computing devices. The high efficiency is also shown in its single-epoch training style, unlike the BP based methods that result in hundreds of epochs or beyond when training a single network. We validate the one-epoch feature as follows. According to the definition in [85], a network being trained for one epoch

5.2. PROPOSED METHOD

Algorithm 5.1 CPNet—training phase

Inputs: input matrix $\mathbf{X}_0$ from the training set, label matrix $\mathbf{Y}$, network length $\mathcal{L}$ or $2M$, activation functions $f_1, \dots, f_{\mathcal{L}-1}$, label encoders $\mathbf{Q}_2, \dots, \mathbf{Q}_{2M}$, and correlation projectors $\mathbf{P}_1, \dots, \mathbf{P}_{2M-1}$.

for $m \leftarrow 1$ **to** M **do**

LE process: obtain encoded label $\tilde{\mathbf{Y}}_{2m}$ with (5.3);

CP process: obtain $\hat{\mathbf{W}}_{2m-1}$ through (5.6) and form activation $\mathbf{X}_{2m-1}$ with (5.5);

LS solution: obtain $\hat{\mathbf{W}}_{2m}$ with (5.8) and form output $\mathbf{Y}_{2m}$ of module l with (5.9);

if $m < M$ **then**

 Calculate the activation $\mathbf{X}_{2m} = f_{2m}(\mathbf{X}_{2m-1} \hat{\mathbf{W}}_{2m-1})$;

end

end

Outputs: $\hat{\mathbf{W}}_1, \dots, \hat{\mathbf{W}}_{\mathcal{L}}$

indicates that the weights in each network module are updated with respect to the number of samples equal to the size of the training set. Our CPNet utilizes exactly one epoch during the training phase according to Algorithm 5.1. That is, for an N -sample dataset, the weights for each module (i.e., $\hat{\mathbf{W}}_{2m-1}$ and $\hat{\mathbf{W}}_{2m}$) are determined using exactly N training samples (i.e., the input $\mathbf{X}_{2m-2} \in \mathbb{R}^{N \times d_{2m-2}}$ and the label $\mathbf{Y} \in \mathbb{R}^{N \times d_y}$) as shown in (5.5), (5.6) and (5.8).

To elaborate, here the one-epoch training between the CPNet and networks trained by BP are compared as follows. Assume that a 2-module network is to be trained on a dataset containing 3 samples, i.e., the inputs $\mathbf{x}_{a0}, \mathbf{x}_{b0}, \mathbf{x}_{c0}$, and the corresponding labels $\mathbf{y}_a, \mathbf{y}_b, \mathbf{y}_c$. In the case of training the network using BP with a mini-batch size of 1, a total of 3 iteration is needed for one epoch. For each iteration, an input (e.g., $\mathbf{x}_{b0}$) is fed to module 1 with its output (e.g., $\mathbf{x}_{b1}$) passing through module 2. The above forward process is then followed by a calculation of loss (e.g., using $\mathbf{x}_{b1}$ and $\mathbf{y}_b$) and gradients, as well as a corresponding weight update. Training the network for one epoch (i.e., 3 iterations) using BP indicates that each module has accessed 3 samples in total (e.g., module 1 has accessed $\mathbf{x}_{a0}, \mathbf{x}_{b0}, \mathbf{x}_{c0}$ to produce $\mathbf{x}_{a1}, \mathbf{x}_{b1}, \mathbf{x}_{c1}$

5.2. PROPOSED METHOD

for module 2). In the case of training the CPNet, according to Algorithm 5.1, all 3 input samples $\mathbf{x}_{a0}$, $\mathbf{x}_{b0}$, and $\mathbf{x}_{c0}$ are fed into module 1 and update its weights. Module 1 then produces $\mathbf{x}_{a1}$, $\mathbf{x}_{b1}$ and $\mathbf{x}_{c1}$ for the training of module 2. In summary, in a single training epoch, each module in CPNet has accessed only 3 samples (e.g., module 1 with inputs samples $\mathbf{x}_{a0}$, $\mathbf{x}_{b0}$, $\mathbf{x}_{c0}$, and module 2 with outputs of module 1, i.e., $\mathbf{x}_{a1}$, $\mathbf{x}_{b1}$, $\mathbf{x}_{c1}$). This is similar to the case for one-epoch training in BP, though the CPNet trains one module after another while the BP updates all the modules in every iteration. Unlike the BP that repeats the one-epoch training for many times, training of the CPNet has ended after finishing the one-epoch training. Hence, the CPNet achieves a one-epoch training style.

As shown in (5.6) and (5.8), the weights in each module are learned through correlation projection and MP inverse respectively, which are one-step learning processes without iteration search. Since these weights are determined with closed-form solutions, there is no convergence issue. In other words, the convergence is guaranteed. Therefore, the CPNet belongs to the category of multilayer analytic learning. The CPNet enables locally supervised learnings like the KARnet and the PILAE without being constrained by invertible activation functions. Also, it can flexibly adjust the network structure through the correction projector, which differs from the PIL and the ANnet that allow only certain restricted structures.

5.2.2 A Multi-exit Prediction Structure for Validation

To make predictions in the validation phase (see Figure 5.1(b)) using a trained CPNet, in correspondence to the LE process, the LD process is proposed which allows each module to give its individual prediction. The CPNet therefore achieves a multi-exit structure for prediction. That is, a K -module CPNet gives K different predictions

5.2. PROPOSED METHOD

(i.e., exits) for the same input. Note that the validation phase is mainly used for determining the network structure, which is different from the testing phase (see Section 5.2.3) that evaluates the performance of the proposed CPNet.

As shown in Figure 5.1(b), at the end of each module, the module output is also connected to a label decoder to generate its own prediction. This is referred to as an *exit* of the network. The label decoder treats the module output as an encoded prediction, and decrypts it by inverting the label encoder. Different from encoders which are chosen randomly, decoders in an M -module CPNet are determined solely based on their encoder counterparts. For an M -module CPNet, the m^{th} ($m \leq M$) exit gives the following prediction:

$$\hat{\mathbf{Y}}_{2m} = f_{2m-1}(\dots f_2(f_1(\mathbf{X}_0 \hat{\mathbf{W}}_1) \hat{\mathbf{W}}_2) \dots) \hat{\mathbf{W}}_{2m} \mathbf{Q}_{2m}^\dagger, \quad m = 1, \dots, M, \quad (5.12)$$

or simply

$$\hat{\mathbf{Y}}_{2m} = \mathbf{X}_{2m-1} \hat{\mathbf{W}}_{2m} \mathbf{Q}_{2m}^\dagger, \quad m = 1, \dots, M \quad (5.13)$$

where $\mathbf{Q}_{2m}^\dagger$ is the label decoder in module m , which is the MP inverse of its corresponding label encoder. The purpose of this LD process is to reverse the LE process in order to make an extra prediction independent of the upper layers. The usefulness of such process can be shown by the following theorem.

Theorem 5.2.1. *Assume that CPNets (1-module, 2-module, ..., M -module) are trained based on the input $\mathbf{X}_0$, label $\mathbf{Y}$, correlation projectors $\mathbf{P}_1, \mathbf{P}_3, \dots, \mathbf{P}_{2m-1}, \dots, \mathbf{P}_{2M-1}$ and label encoders $\mathbf{Q}_2, \mathbf{Q}_4, \dots, \mathbf{Q}_{2m}, \dots, \mathbf{Q}_{2M}$. Then, if the label encoder $\mathbf{Q}_{2m}$ has full row rank, the m^{th} ($m < M$) exit of an M -module CPNet gives the same prediction as that of an m -module CPNet.*

5.2. PROPOSED METHOD

Proof. Let the weights of the m -module and the M -module CPNets be $\{\hat{\mathbf{W}}_1, \hat{\mathbf{W}}_2, \dots, \hat{\mathbf{W}}_{2m-1}, \hat{\mathbf{W}}_{2m}\}$ and $\{\hat{\mathbf{W}}_1, \hat{\mathbf{W}}_2, \dots, \hat{\mathbf{W}}_{2m-1}, \hat{\mathbf{W}}_{2m}, \dots, \hat{\mathbf{W}}_{2M}\}$ respectively. Note that they share the same weights in the first $2m - 1$ layers due to performing identical computations.

Let $\mathbf{X}_{2m-1} = f_{2m-1}(\dots f_2(f_1(\mathbf{X}_0 \hat{\mathbf{W}}_1) \hat{\mathbf{W}}_2) \dots \hat{\mathbf{W}}_{2m-1})$ be the activation of layer $2m - 1$. The prediction of the m -module CPNet is given by

$$\hat{\mathbf{Y}}_{2m} = \mathbf{X}_{2m-1} \hat{\mathbf{W}}_{2m} = \mathbf{X}_{2m-1} \mathbf{X}_{2m-1}^\dagger \mathbf{Y}, \quad m = 1, \dots, M. \quad (5.14)$$

On the other hand, according to (5.13), the m^{th} exit of the M -module CPNet gives the following prediction:

$$\hat{\mathbf{Y}}_{2m} = \mathbf{X}_{2m-1} \hat{\mathbf{W}}_{2m} \mathbf{Q}_{2m}^\dagger, \quad m = 1, \dots, M. \quad (5.15)$$

According to (5.8) and (5.3), $\hat{\mathbf{W}}_{2m}$ is determined by

$$\hat{\mathbf{W}}_{2m} = \mathbf{X}_{2m-1}^\dagger \tilde{\mathbf{Y}}_{2m} = \mathbf{X}_{2m-1}^\dagger \mathbf{Y} \mathbf{Q}_{2m}, \quad m = 1, \dots, M$$

which can therefore rewrite (5.15) as

$$\hat{\mathbf{Y}}_{2m} = \mathbf{X}_{2m-1} \mathbf{X}_{2m-1}^\dagger \mathbf{Y} \mathbf{Q}_{2m} \mathbf{Q}_{2m}^\dagger, \quad m = 1, \dots, M. \quad (5.16)$$

If the label encoder $\mathbf{Q}_{2m}$ has full row rank, we have $\mathbf{Q}_{2m} \mathbf{Q}_{2m}^\dagger = \mathbf{I}$. The right hand side of (5.16) can therefore be simplified as $\mathbf{X}_{2m-1} \mathbf{X}_{2m-1}^\dagger \mathbf{Y}$ which is identical to (5.14). That is, the m^{th} exit of an M -module CPNet and an m -module CPNet give identical prediction.

□

5.2. PROPOSED METHOD

Theorem 5.2.1 demonstrates the usefulness of CPNet’s multi-exit structure. Apart from the prediction given by the last layer, an M -module CPNet automatically obtains the exact predictions given by its sub-networks, i.e., the 1-module, 2-module, $\dots$, $(M - 1)$ -module networks, as long as the label encoders are of full row rank. Since the label encoders are determined by drawing random numbers from the normal distribution, these encoders are likely to have full row ranks. This could be useful for determining the depth of the network. One can simply train one deeper network to see which depth gives the best performance. This is much more effective than training multiple networks of various depths. The multi-exit prediction structure is summarized in Algorithm 5.2.

Algorithm 5.2 CPNet—Making predictions in the validation process

Inputs: input matrix $\mathbf{X}_0$ from the validation set, activation functions $f_1, \dots, f_{\mathcal{L}-1}$, correlation projectors $\mathbf{P}_1, \dots, \mathbf{P}_{2M-1}$, label decoders $\mathbf{Q}_2^\dagger, \dots, \mathbf{Q}_{2M}^\dagger$, and weights $\{\hat{\mathbf{W}}_1, \dots, \hat{\mathbf{W}}_{2M}\}$ trained using Algorithm 5.1 on the training set.

for $m \leftarrow 1$ **to** M **do**

 Obtain module output $\mathbf{Y}_{2m}$ using (5.5) and (5.9) such that: $\mathbf{Y}_{2m} = f_{2m-1}(\mathbf{X}_{2m-2} \hat{\mathbf{W}}_{2m-1}) \hat{\mathbf{W}}_{2m}$;

if $l < M$ **then**

 Obtain the prediction $\hat{\mathbf{Y}}_{2m}$ at the m^{th} exit using (5.13);
 Calculate the activation $\mathbf{X}_{2m} = f_{2m}(\mathbf{X}_{2m-1} \hat{\mathbf{W}}_{2m-1})$;

else

 Obtain the prediction at the last layer: $\hat{\mathbf{Y}}_{2M} = \mathbf{X}_{2M-1} \hat{\mathbf{W}}_{2M}$;

end

end

Outputs: $\hat{\mathbf{Y}}_2, \dots, \hat{\mathbf{Y}}_{2M}$ for structure selection purposes.

5.2.3 The Testing Phase

As illustrated in Figure 5.2, the network structure is explored and selected by conducting a validation process (e.g., using k -fold cross validation [84]) on the training

5.2. PROPOSED METHOD

set. Based on the structure selected (e.g., the red dash box in Figure 5.2(a)) from the validation phase, performance of the CPNet is evaluated using the unseen test set after being retrained on the entire training set. Note that reported performance is based on the prediction output from the final layer of the selected network (see Figure 5.2(b)). Specifically, suppose the validation recommends that the CPNet has an m -module structure. After being retrained on the training set with a set of trained weights $\{\hat{\mathbf{W}}_1, \hat{\mathbf{W}}_2, \dots, \hat{\mathbf{W}}_{2m-1}, \hat{\mathbf{W}}_{2m}\}$ ($\hat{\mathbf{W}}_{2m} \in \mathbb{R}^{d_{2m-1} \times d_y}$ is used to distinguish the weight at the final layer from those hidden layer weights $\hat{\mathbf{W}}_{2m} \in \mathbb{R}^{d_{2m-1} \times d_{2m}}$ used in the validation phase), the prediction output during the testing phase is obtained as

$$\hat{\mathbf{Y}} = f_{2m-1}(\dots f_2(f_1(\mathbf{X}_t \hat{\mathbf{W}}_1) \hat{\mathbf{W}}_2) \dots \hat{\mathbf{W}}_{2m-1}) \hat{\mathbf{W}}_{2m}, \quad m = 1, \dots, M, \quad (5.17)$$

which utilizes the unseen test data $\mathbf{X}_t$. Neither multi-exit structure nor ensemble-like method has been used during the testing phase. This is to provide a fair performance comparison among different methods by ensuring networks trained by different methods having the same MLP structure with only one prediction output during testing. The detailed steps for the validation and testing can be found in the subsequent experiment section.

In general, machine learning algorithms require a validation process to determine their structures, which might need to train the same network structure multiple times for evaluation. The CPNet would seemingly need several visits (i.e., multiple epochs) of the training set. However, such a process does not contradict the CPNet’s one-epoch feature. Take the k -fold cross validation as an example, for each of the k folds, one unique CPNet is trained. The cross validation in fact trains k different CPNets of the same structure. That is, training one unique CPNet still requires one epoch only, which supports the claim of one-epoch learning.

5.2. PROPOSED METHOD

Figure 5.2: An illustration of (a) validation phase determining the network structure (highlighted in the red dash box), and (b) testing phase evaluating the performance of the selected network. In this illustration, three network structures are evaluated with “M=m, H=h” for hidden neuron sizes $h=100, 300, 1000$ on modules $m=1, \dots, 5$ in order to select the best validated network structure using only the training set. The network structure with $m=3$ and $h=300$ has been selected in view of its highest validation accuracy of 90%.

5.2.4 Space Complexity Analysis

Here a brief analysis of space complexity for CPNet is conducted. For each module, the major space complexity happens in the CP layer in (5.6) and LS layer in (5.8). Specifically, the complexity of (5.6) is $\mathcal{O}\{Nd_{2m-2}d_{2m-1}\}$, and the complexity of (5.8) is $\max\{\mathcal{O}\{Nd_{2m-1}^2\}, \mathcal{O}\{Nd_{2m-1}d_{2m}\}\}$ where $\mathcal{O}\{Nd_{2m-1}^2\}$ is from the calculation of MP inverse and $\mathcal{O}\{Nd_{2m-1}d_{2m}\}$ is from the matrix multiplication. That is, the complexity is proportional to the hidden sizes and the sample size. Specifically, if the sample size is very large and dominates the complexity, recursive methods can be adopted to reduce the complexity at the expense of spending more time. This will be further investigated in Chapter 6.

5.3 Experiments

In this section, the empirical performance of the proposed CPNet and its training efficiency are studied through classification tasks. Experiments are conducted on various benchmark datasets, including those from the UCI repository¹, the ORL face², and the MNIST³ datasets. The experiments are run using a CPU (E3-1226 v3) with 16GB RAM. We compare the proposed CPNet with several competing methods, including KARnet [18], ANnet [65], RBM-GI [86], PILAE [61], RBF [22], and PNN [87]. The commonly used BP [67] is also experimented for benchmarking purpose. The BP has been used to train the MLP (MLP-BP) as well as the convolutional neural network (CNN), i.e., LeNet-5 [2]. In particular, only the CNN LeNet-5's performance on the MNIST dataset is reported as this type of network is mainly for image-based datasets with a very large number of training samples.

For the proposed CPNet, the ReLU function [88] is used as the activation function. In the following experiments, instead of exhausting every possible option in exploring the multilayer network structure, the hyperparameter search is simplified by having an identical number of neurons in each hidden layer. In such a way, there is only one hyperparameter (i.e., hidden neuron size in each hidden layer) to tune.

To determine the network structure, a 10-fold cross validation based on a given training set is conducted. A relatively deep CPNet, i.e., a 5-module (consisting of 10 layers) one, is trained to determine the hidden neuron size and the network depth according to the particular exit that gives the best validation performance of the training set. Specifically, a 10-fold cross validation is performed for each given candidate structure on the training set and compare the validation performance produced

¹<https://archive.ics.uci.edu/ml/datasets.html>

²http://nimfa.biolab.si/nimfa.examples.orl_images.html

³<http://yann.lecun.com/exdb/mnist/>

5.3. EXPERIMENTS

by all the exits and by all the candidates. For instance, as illustrated in Figure 5.2, the goal is to select a hidden neuron size from 100, 300 and 1000. Towards this goal, a 10-fold cross validation is performed for each of the three 5-module networks with 100, 300 and 1000 hidden sizes. Within the 10-fold cross validation for each network candidate, 5 accuracies from 5 exits for each fold are obtained. For each exit the accuracy is averaged across all the 10 folds. The identical process continues for all the network candidates. We then select the structure with an exit that gives the best average validation accuracy. The example in Figure 5.2 recommends a 3-module structure (marked by the red dash box) with a hidden size of 300 as the 3rd exit gives the best validation accuracy of 90%. Subsequently, the selected 3-module network is re-trained using (5.6) and (5.8) on the entire training set, and its testing performance is reported on the testing set using (5.17) obtained from the final layer.

Particularly, for datasets that have their own testing sets, 10 trials of the retraining-and-testing process are repeated utilizing different random seeds for conducting CP processes and building label encoders, and the average accuracy of these 10 trials is reported. For datasets without their own testing sets, for each trial a testing set (10% of total data) is randomly partitioned from the entire dataset, and the aforementioned retraining-and-testing process is performed, with the performance reported by the average accuracy. This 10-trial testing is to obtain a performance with strong statistical confidence. The processes of tuning (of network structure using only the training set) based on the 10-fold cross validation, and testing based on the 10-trial performance averaging have been utilized in all the following experiments.

Among the compared methods, the PNN is implemented with *NeuPy*⁶. The implementation of RBF is based on the PyTorch code⁷ with a Gaussian kernel. The

⁶<http://neupy.com/apidocs/neupy.algorithms.rbfn.pnn.html>.

⁷<https://github.com/JeremyLinux/PyTorch-Radial-Basis-Function-Layer>.

5.3. EXPERIMENTS

Table 5.2: UCI dataset specifications.

Dataset	# Attributes	# Samples	# Classes
Letter	16	20000	26
Optdigits	64	5620	10
Pendigits	16	10992	10
Balance	4	625	3
Waveform	21	5000	3
Mushroom	22	8124	3
Iris	4	150	3
Shuttle	9	43500	7
Sonar	60	208	2

MLP-BP is implemented based on the PyTorch code⁸, and a learning rate of 0.01 is used with an SGD optimizer and the network is trained with a batch size of 128 for 50 epochs.

5.3.1 UCI Datasets

First, several datasets from the UCI repository (see their specifications in Table 5.2) are considered for experimentation. These datasets include two categories. The first category contains *Optdigits*, *Pendigits*, and *Shuttle* that give their own training and testing sets, and the second category includes *Letter*, *Balance*, *Waveform*, *Mushroom*, *Iris* and *Sonar* which do not. For the second category of datasets, they are randomly split into a training set (90%) and a testing set (10%) to facilitate the 10-fold cross validation. During the validation, the CPNet structure is evaluated by exploring the hidden neuron size from 50 to 10000.

The results in Table 5.3 show that the proposed CPNet outperforms the compared methods on 5 of the UCI datasets, including the Balance, Waveform, Iris, Shuttle, and Sonar datasets. For the other 4 datasets, the CPNet gives very close results to those

⁸<https://github.com/iam-mhaseeb/Multi-Layer-Perceptron-MNIST-with-PyTorch>.

5.3. EXPERIMENTS

Table 5.3: Average testing accuracies (%) of compared methods on UCI datasets.

		Letter	Optdigits	Pendigits	Balance	Waveform	Mushroom	Iris	Shuttle	Sonar
KARnet (2-layer) [89]		88.99	97.25	97.13	88.47	83.20	95.56	96.40	96.74	74.25
KARnet (3-layer) [89]		94.32	97.17	96.22	88.77	83.07	95.69	97.47	96.67	71.75
KARnet (4-layer) [89]		94.12	96.96	96.05	90.92	83.27	95.71	96.53	95.52	72.70
ANnet (2-layer) [65]		91.79	96.82	99.54	91.43	82.90	95.71	97.13	96.78	73.60
ANnet (3-layer) [65]		69.29	99.06	98.36	90.98	82.17	95.62	94.00	96.85	73.40
RBM-GI [86]		74.45	96.83	95.94	85.82	84.00	100.0	-	91.55	-
RBF [22]		80.00	95.80	95.60	85.71	85.20	100.0	100.0	94.53	78.09
PNN [87]		96.15	97.99	97.74	87.30	76.60	100.0	98.80	97.10	71.15
MLP-BP [67]		95.94	94.66	95.05	90.05	82.20	100.0	100.0	96.56	77.75
CPNet (5-module validation)	1 st exit	69.97	98.33	98.29	91.46	84.69	99.67	92.31	95.58	76.24
	2 nd exit	93.92	99.24	99.09	92.88	84.53	99.64	98.46	96.92	76.64
	3 rd exit	93.92	99.17	99.12	92.17	84.11	99.66	98.46	97.09	76.65
	4 th exit	93.95	99.16	99.08	91.45	84.18	99.69	98.46	97.08	72.91
	5 th exit	93.72	99.13	99.09	92.17	84.24	99.66	98.46	97.02	73.31
CPNet (testing)		94.73	97.24	97.19	93.33	86.18	99.67	100.0	97.20	78.10

1. The results for KARnet are reported from [89]. For ANnet, the numbers of Letter, Optdigits and Pendigits are reported from [65] while the results for the other datasets are reported after exploring the depth from 1 to 5. In addition, the results for RBM-GI are obtained from [86]. The MLP-BP experiments are reported after exploring the hidden neuron size and depth, from 50 to 10000 and from 1 to 5, respectively. For the RBF method, the standard deviation and the number of centers are explored from 0.1 to 10 and from 50 to 10000, respectively. Results for PNN are reported after exploring the standard deviation from 0.1 to 10.
2. The best testing results are marked with **black** boldface numbers. The exit of CPNet that gives the best validation accuracies are marked **blue**. The testing results are reported by the final exit of the retrained CPNets with lengths consistent with the corresponding exits. For instance, on the Pendigits dataset, the accuracy is reported by the 3-module CPNet that gives 97.19%.

from the competing methods (e.g., 99.67% vs. 100.0% on the Mushroom dataset).

Table 5.3 also reveals that the CPNet benefits tremendously from its multi-exit structure. The compared methods need to train multiple networks to determine the network depth (see details in [18, 65, 86]). Our method straightforwardly trains a deep network (5-module or equivalently 10-layer network) and selects the depth that outputs the best validation accuracy. For instance, during the 10-fold cross validation, after training a 5-module network on Sonar dataset, without further exploration, it is immediately discovered that the network with 3 modules is a good candidate. This further simplifies the hyperparameter tuning, and is hence more efficient.

Another important observation is that the overall performance of the multilayer CPNet is better than its 1-module shallow structure. During the validation, a 1-module CPNet on Letter dataset gives an accuracy of only 69.97%, but an additional module improves the result dramatically by about 24%. Other datasets do not demon-

Figure 5.3: Samples from the ORL face recognition dataset.

strate such obvious increase. This might be due to the fact that the Letter dataset is relatively complex, so the CP process in the first module of CPNet cannot sufficiently extract the feature information. The accuracy is improved greatly with more modules because the subsequent modules pick up further information for discrimination.

5.3.2 ORL Face Recognition

The ORL face dataset contains 400 face images taken from 40 persons. Each person has 10 image samples containing some variations of facial expressions and facial orientations. Our task is to classify a given face into one of the 40 classes (40 persons). Some sample images are shown in Figure 5.3. Each image is reshaped into the size of 32×32 . We randomly select 90% samples from each class as training data, and the rest 10% as testing data.

As shown in Table 5.4, the 10-fold cross validation recommends the third exit of the network with a hidden neuron size of 1800. Using this structure, the proposed CPNet achieves an accuracy of 98.75%, which is the highest among the compared methods. On this dataset, the CPNet gives a good performance with a relatively narrow structure (1800 hidden neuron size) among the analytic learning based methods. For instance, the RBM-GI requires up to 10000 neurons in the 4th layer to obtain a 97.50% accuracy.

5.3. EXPERIMENTS

Table 5.4: Performance on ORL face dataset (N.A. means *not applicable*).

		Structure	Test Acc (%)
KARnet [89]		-3000-	90.00
ANnet [65]		-1024-	92.50
RBM-GI [86]		-2500-2500-2500-10000-	97.50
RBF [22]		-1000-	95.00
PNN [87]		N.A.	95.00
MLP-BP [67]		-1000-	95.00
CPNet (5-module, validation)	1 st exit	-1800-	95.28
	2 nd exit	-{1800}- $\times$ 3	98.06
	3 rd exit	-{1800}- $\times$ 5	98.33
	4 th exit	-{1800}- $\times$ 7	98.33
	5 th exit	-{1800}- $\times$ 9	98.33
CPNet (testing)		{1800}- $\times$ 5	98.75

1. The RBM-GI's recognition performance and structure are reported from paper [86].
2. The structures (2-layer) for KARnet and MLP-BP are found by those that give the best testing accuracies after exploring the hidden layer size from 500 to 5000. The result for ANnet is reported after examining the depth from 1 to 5. The structures of RBF and PNN are determined in the same way as those for the UCI case.
3. Architecture $-\{.\}-\times K$ indicates a $(K+1)$ -layer network. For instance, $-\{100\}-\times 3$ is a 4-layer network of "-100-100-100-" structure.

Figure 5.4: Samples from the MNIST dataset.

5.3. EXPERIMENTS

Table 5.5: Performance on MNIST dataset.

Methods	Architecture	Test Acc (%)
KARnet [18]	-1000-	91.24
KARnet [18]	-5000-	92.39
KARnet [18]	-10000-	92.44
KARnet [18]	-{10000}- $\times 2$	91.18
ANnet [65]	-1000-	94.37
ANnet [65]	-10000-	97.85
ANnet [65]	-27000-	98.24
RBM-GI [86]	-700-500-500-10000-	97.58
PILAE [61]	-	96.92
RBF [22]	-8000-	97.88
PNN [87]	N.A.	96.91
MLP-BP [67]	-1000-	98.25
MLP-BP [67]	-{1000}- $\times 2$	98.04
MLP-BP [67]	-10000-	98.42
MLP-BP [67]	-{10000}- $\times 2$	98.31
CNN-BP (LeNet-5) [2]	-CONV-84-*	99.05*
CPNet (5-module, validation)	1 st exit	-8000- 97.76
	2 nd exit	-{8000}- $\times 3$ 97.88
	3 rd exit	-{8000}- $\times 5$ 97.90
	4 th exit	-{8000}- $\times 7$ 97.88
	5 th exit	-{8000}- $\times 9$ 97.87
CPNet (5-module, testing)	-{8000}- $\times 5$	97.91
CPNet (5-module, validation)	1 st exit	-15000- 97.80
	2 nd exit	-{15000}- $\times 3$ 98.13
	3 rd exit	-{15000}- $\times 5$ 98.13
	4 th exit	-{15000}- $\times 7$ 98.14
	5 th exit	-{15000}- $\times 9$ 98.13
CPNet (5-module, testing)	-{15000}- $\times 7$	98.50

1. The structures of RBF and PNN are determined in the same way as that for the UCI case.
2. *We have also included the LeNet-5 as a baseline for CNN. Its structure and performance are reported from the LeNet-5 paper [2]. Specifically, the “CONV” in LeNet-5 contains 3 convolution layers of kernel 5×5 with 6, 16 and 120 channels respectively, where the first two convolution layers are followed by a stride 2 average pooling. The LeNet-5 used the *tanh* activation function.

5.3.3 MNIST Dataset

The MNIST dataset contains images with 28×28 pixels for recognizing digits from 0 to 9. Some samples are shown in Figure 5.4. This dataset has 60000 and 10000 images for training and testing respectively. We apply 10-fold cross validation on the training set with a search range of between 1000 to 20000 neurons. The testing accuracy is reported in terms of the average of accuracies on the testing set for 10 trials utilizing different random seeds for conducting CP processes and building label encoders. That is, the selected network is retrained on the training set and measure the testing accuracy on the testing set. We then repeat the retaining-and-testing for 10 times utilizing different random seeds during the training stage. The final testing accuracy is reported by averaging the accuracies from these 10 trials for statistical confidence.

The results are shown in Table 5.5. From the reported numbers, the KARnet is not able to reach beyond 93%. The RBM-GI, PILAE, RBF, and PNN produce slightly worse results (97.58%, 96.92%, 97.88%, and 96.69% respectively) than that of the BP (98.42%) while the ANnet gives a comparable one (98.24%) using a shallow structure with 27000 hidden neurons. For the proposed CPNet, the 10-fold cross validation utilizing only the training set determines the best network structure as “ $-\{15000\} - \times 7$ ”. According to the testing performance, the proposed CPNet is able to slightly outperform the BP by achieving a 98.50% classification accuracy. This result is obtained with a 4-module CPNet. As a baseline algorithm, the LeNet-5 [2]—a CNN structure—is also included, which has been shown to consistently outperform MLPs on image datasets. This demonstrates that, although the CPNet is of MLP structure, it is almost as powerful as a CNN (99.05%) on the MNIST dataset.

Unlike the BP, analytic learning methods tend to give improved results with wider

5.3. EXPERIMENTS

network structures. This can be supported by the reported performances in Table 5.5. For instance, the KARnet improves its performance by 1.2% after increasing the hidden neuron counts from 1000 to 10000. The ANnet achieves a 3.87% gain after tuning the width from 1000 to 27000. The RBM-GI also admits similar numerical patterns (see Section V-B in [86]). Our proposed CPNet seems to inherit this property. In order to observe the impact of hidden node size on the accuracy, a CPNet with a narrower hidden size of 8000 neurons (see Table 5.5) has been experimented. The result shows that a gain of about 0.6% of accuracy is obtained if the hidden size is increased from 8000 to 15000.

5.3.4 Runtime Comparison

We now evaluate the efficiency of the proposed CPNet. This is reported by the training time of the compared methods for training a 2-layer network with 1000 hidden neurons on the MNIST dataset. Note that the PNN and LeNet-5 are not included as they do not have the same structure.

As shown in Figure 5.5, training the RBF network requires the longest time due to its time-consuming kernel computation. The proposed CPNet spends 3.139 seconds to complete the training on the MNIST dataset. This is the second best result among the compared methods. The ANnet only needs 2.476 seconds as it directly treats the features as its weights in hidden layers. In terms of generalization performance, the ANnet gives slightly worse results than those of CPNet. In addition, ANnet has a structure restriction as the sizes in hidden layers cannot be chosen arbitrarily [65]. The KARnet also obtains very promising acceleration result with 3.481 seconds. In comparison with CPNet regarding generalization, the KARnet’s performance has been shown to be less ideal (e.g., 92.44% for KARnet v.s. 98.5% for CPNet on MNIST).

5.3. EXPERIMENTS

Among analytic learning methods, the CPNet achieves promising generalization and speed results without structural constraints.

In general, the analytic learning based techniques (e.g., KARnet, ANnet and CPNet) give significantly higher training efficiencies compared with the commonly used BP method. The iterative-search nature of the BP method usually results in a training with hundreds of epochs, which inevitably leads to longer execution time. For instance, in the MNIST experiment, a 50-epoch BP spends 211.5 seconds, which is significantly longer than those of the other compared methods (e.g., CPNet for 3.139 seconds). This empirical evidence further supports the motivation towards developing methods for training the neural networks analytically.

The datasets adopted in the experiments are relatively small as the analytic learning methods are restricted to MLP only at the current stage. Hence, the feature size and sample size are relatively small. However, if needed, we believe that the CPNet can be scaled to considerable large features (e.g., more than 15000 nodes in hidden layers) as long as the machine has enough memory to conduct MP inverse of matrix with the size corresponding to such a feature size. Also, the sample size can be very large (e.g., more than 6000 samples from the MNIST case) because we can adopt recursive-based methods to reduce the memory consumption (see Chapter 6 for instance).

Note that, although the CPNet is mainly designed to perform classification tasks, it should be equally effective for regression tasks since the our solution is based on solving LS problems. However, there could be difficulties when dealing with regression problems involving loss functions beyond the linear ones. This difficulty applies in the classification problem as well and will be investigated in the future.

In summary, results in the experiments have revealed that:

5.3. EXPERIMENTS

Figure 5.5: Training time (seconds) comparison among the compared methods.

1. The CPNet demonstrates competitive results in classification tasks. It outperforms the compared methods on many datasets, including several datasets in the UCI repository, the ORL face dataset, and the MNIST dataset.
2. We obtain empirical evidence indicating that extra modules could help the proposed CPNet to achieve better performance. For instance, on the Letter dataset a 2-module CPNet improves the accuracy by 24% compared with its 1-module counterpart.
3. We show that the CPNet can train the network much faster than the commonly used BP method. As an example of training the MNIST dataset, the CPNet trains the network within 3.2 seconds while the BP needs more than 200 seconds.

5.4 Summary

In this chapter, the CPNet, a multilayer network that learns its parameters analytically, has been introduced. Essentially, the CPNet consisted of multiple 2-layer modules. To achieve analytic learning, the label was first encoded into the hidden layer through a label encoding process. Subsequently, in each module, the first layer extracted features using a correlation projection process, while the second layer mapped the features onto the encoded label by solving the least squares problem. The label encoding process, which allowed a multi-exit feature during the validation phase, was invertible. Our empirical results showed that the CPNet performed favorably compared with several competing state-of-the-arts. The experiments also indicated better performance with deeper networks on several datasets. In view of the computational efficiency, the training time of CPNet was observed to be significantly faster (e.g., $60\times$ faster on the MNIST dataset) than the commonly used backpropagation.

Chapter 6

Block-wise Recursive Moore-Penrose Inverse for Network Learning

The MP inverse is known to be a key component for analytic learning methods. Implementing the MP inverse in a batch-wise manner could induce overwhelming memory demand. Such a memory issue can be circumvented by implementing the MP inverse recursively [39, 40]. However, existing methods are sample-wise where the training samples are processed one-by-one in each iteration, which leads to significantly long execution time. This has motivated us to develop a new formulation beyond the sample-wise capacity to improve the computational efficiency while preserving the low-memory property. In this chapter, the block-wise MP inverse formulation (BRMP) is proposed, which is a block-wise recursive implementation of the LS solution based on the MP inverse. The BRMP allows the LS solution to run with low-memory consumption while preserving its computational effectiveness, which can

6.1. PRELIMINARIES

Table 6.1: Table of Notations.

Notations	Description
$\omega/\mathbf{W}$	Weight vector/matrix.
$\hat{\omega}^{(t)}/\hat{\mathbf{W}}^{(t)}$	Weight vector/matrix estimation at step t .
$\mathbf{H}$	Data matrix.
$\mathbf{H}_t$	Accumulated data matrix at step t .
$\mathcal{H}_t$	Arriving Data matrix at step t .
$\mathbf{z}/\mathbf{Z}$	Label vector/matrix.
$\mathbf{z}_t/\mathbf{Z}_t$	Accumulated label vector/matrix at step t .
$\mathbf{z}_t/\mathbf{Z}_t$	Arriving label/matrix vector at step t .
$\mathbf{G}_t$	Gain matrix at step t .
$\mathbf{C}_t$	Residual matrix at step t .
$\mathbf{Q}_t$	Kernel projector at step t .

be directly applied to the existing analytic learning methods for memory reduction.

The rest of this chapter is presented as follows. Section 6.1 revisits the background knowledge prior to further development of the proposed method. Section 6.2 elaborates the derivation of the BRMP. In addition, the applications of BRMP in network learning, including shallow networks and multilayer networks, are presented in Section 6.3. The claims made in this chapter are then validated in Section 6.4 through extensive experiments. The chapter is then summarized in Section 6.5.

6.1 Preliminaries

In this section, the basic linear regression problem with its connection to the MP inverse is revisited. For convenience, frequently used notations in this chapter are presented in Table 6.1.

6.1.1 The Linear Regression Problem

Given a dataset $\{h_{i1}, h_{i2}, \dots, h_{iK}, z_i\}_{i=1}^N$ with N samples, each of which has K features, where h_{ik} is the k^{th} feature in the i^{th} sample, and z_i is the label of the i^{th} sample. Assume that the features follow a linear connection to the label, such that

$$z_i = w_1 h_{i1} + w_2 h_{i2} + \dots + w_K h_{iK}.$$

This can be formulated into a matrix form:

$$\mathbf{z} = \mathbf{H}\mathbf{w} \tag{6.1}$$

where $\mathbf{w} \in \mathbb{R}^K$, $\mathbf{z} \in \mathbb{R}^N$ and $\mathbf{H} \in \mathbb{R}^{N \times K}$ are the weight vector, the label vector, and the data matrix respectively, constructed via

$$\mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_K \end{bmatrix}, \mathbf{z} = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_N \end{bmatrix}, \mathbf{h}_i = \begin{bmatrix} h_{i1} \\ h_{i2} \\ \vdots \\ h_{iK} \end{bmatrix}, \mathbf{H} = \begin{bmatrix} \mathbf{h}_1^T \\ \mathbf{h}_2^T \\ \vdots \\ \mathbf{h}_N^T \end{bmatrix}.$$

The formulation in (6.1) forms a linear system which can be solved through minimizing the residual:

$$\underset{\mathbf{w}}{\operatorname{argmin}} \quad \|\mathbf{z} - \mathbf{H}\mathbf{w}\| \tag{6.2}$$

6.1. PRELIMINARIES

where $\|\cdot\|$ represents the l_2 -norm of a matrix/vector. An LS solution to (6.2) is given by:

$$\hat{\mathbf{w}} = \mathbf{H}^\dagger \mathbf{z}. \quad (6.3)$$

This solution is optimal as the residual

$$r = \|\mathbf{z} - \mathbf{H}\hat{\mathbf{w}}\| \quad (6.4)$$

is minimum for all possible $\hat{\mathbf{w}} \in \mathbb{R}^K$. The developed solution takes the entire dataset (N samples) to calculate the weights directly. (6.3) is referred to as a *batch-wise* solution to the linear regression problem.

6.1.2 Moore-Penrose Inverse

The MP inverse (also called the pseudoinverse) of a matrix is a widely known generalization of the matrix inverse. Its definition for real numbers¹ is as follows.

Definition 6.1.1. [90] *Let $\mathbf{A} \in \mathbb{R}^{m \times n}$. Its MP inverse $\mathbf{A}^\dagger \in \mathbb{R}^{n \times m}$ is a unique matrix that meets the following four conditions: 1) $\mathbf{A}\mathbf{A}^\dagger\mathbf{A} = \mathbf{A}$, 2) $\mathbf{A}^\dagger\mathbf{A}\mathbf{A}^\dagger = \mathbf{A}^\dagger$, 3) $(\mathbf{A}\mathbf{A}^\dagger)^T = \mathbf{A}\mathbf{A}^\dagger$, and 4) $(\mathbf{A}^\dagger\mathbf{A})^T = \mathbf{A}^\dagger\mathbf{A}$.*

As shown in Definition 6.1.1, the MP inverse $\mathbf{A}^\dagger$ exists and is unique for a given arbitrary matrix $\mathbf{A}$. This definition is more general than the commonly used left inverse $(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T$ and the right inverse $\mathbf{A}^T(\mathbf{A}\mathbf{A}^T)^{-1}$, both of which are only applicable with a full-rank matrix $\mathbf{A}$. The MP inverse also constructs an optimal solution that minimizes the LS error in linear problems [33, 90] such as (6.2).

¹The MP inverse's definition includes that for the complex numbers, but in this thesis only real numbers are considered.

6.1.3 The Memory Growing Problem and Recursive Methods

The solution in (6.3) to the minimization problem (6.2) is compact and easy to interpret. However, there is an obvious downside for the batch-wise computation. This is because a direct MP inverse followed by a matrix multiplication consumes the amount of memory proportional to the size N (for $N > K$) of a dataset. For large N , equation (6.3) could suffer from OOM. Although sample-wise recursive methods, such as [40,91], have been proposed to tackle the memory issue, they achieve this goal at the expense of much longer computational time than the batch-wise solution.

In the following sections, the memory issue caused by a large dataset can be avoided by formulating the MP inverse in a block-wise recursive manner. The formulation allows the weights to be updated using an arbitrary block of the dataset in each iteration. It is shown to achieve a much more affordable memory-speed trade-off than that of the sample-wise recursive methods.

6.2 Block-wise Recursive Moore-Penrose Inverse

In this section, the BRMP is formulated to solve problem (6.2) through a block partitioning of the MP inverse. The formulation is equivalent to the batch-wise LS as it requires neither assumption nor approximation in the derivation. The proposed BRMP is of a switching structure. Such a structure also establishes a relationship between the proposed method and the well-known RLS.

Prior to further developments, the data matrix $\mathbf{H}$ and the label vector $\mathbf{z}$ are split into T arbitrary blocks. Each block ($\mathcal{H}_t \in \mathbb{R}^{n_t \times K}$ from $\mathbf{H}$ and $\mathbf{z}_t \in \mathbb{R}^{n_t}$ from $\mathbf{z}$) contains n_t rows such that $\sum_{t=1}^T n_t = N$. Note that $\mathcal{H}_t$ and $\mathbf{z}_t$ could have various sizes for different t . The objective of the proposed BRMP is to update the parameters

using one block only instead of taking $\mathbf{H}$ and $\mathbf{z}$ all at once. Accordingly, $\mathbf{H}_t$ and $\mathbf{z}_t$ are referred to as the *accumulated data* and *accumulated label* at step t :

$$\mathbf{H}_t = [\mathbf{H}_{t-1}^T \quad \mathcal{H}_t^T]^T, \quad \mathbf{z}_t = [\mathbf{z}_{t-1}^T \quad \mathbf{z}_t^T]^T \quad (6.5)$$

with $\mathbf{H}_1 = \mathcal{H}_1$ and $\mathbf{z}_1 = \mathbf{z}_1$. In such case, $\mathcal{H}_t$ and $\mathbf{z}_t$ may be termed the *arriving data* and *arriving label* at step t .

The proposed BRMP examines the arriving data $\mathcal{H}_t$ for its dependency upon the accumulated data $\mathbf{H}_{t-1}$ based on the *residual* matrix $\mathbf{C}_t$ (defined in (6.12)) and the *kernel projector* $\mathbf{Q}_{t-1}$ (defined in (6.17)). The residual matrix $\mathbf{C}_t$ extracts $\mathcal{H}_t$'s linearly independent component with respect to the accumulated data $\mathbf{H}_{t-1}$ while the $\mathbf{Q}_{t-1}$ is an orthogonal projector that projects a matrix onto the kernel of the accumulated data $\mathbf{H}_{t-1}$. Note that $\mathbf{Q}_{t-1}$ instead of $\mathbf{Q}_t$ is adopted here because the projector at step $t-1$ is closely connected to the residual matrix at step t (see their connection given in (6.21) later). Figure 6.1 shows that such examination results in a switching structure among three scenarios based on whether $\mathbf{Q}_{t-1} = \mathbf{0}$ or $\mathbf{C}_t = \mathbf{0}$. Scenario I corresponds to the general formulation of BRMP (see Theorem 6.2.2) with $\mathbf{C}_t \neq \mathbf{0}$ and $\mathbf{Q}_{t-1} \neq \mathbf{0}$. Scenario II refers to the case with $\mathbf{C}_t = \mathbf{0}$ but $\mathbf{Q}_{t-1} \neq \mathbf{0}$ (see Corollary 6.2.1). This means that the arriving data $\mathcal{H}_t$ is fully dependent on the accumulated data. Scenario III (see Corollary 6.2.2) corresponds to $\mathbf{Q}_{t-1} = \mathbf{0}$, indicating that the kernel of $\mathbf{H}_{t-1}$ is a null space and thus $\mathbf{H}_{t-1}$ is of full column rank. In particular, this scenario links the well-known RLS to the proposed BRMP.

Figure 6.1: The structure of the proposed BRMP, which switches among Scenarios I-III (corresponding to Theorems 6.2.2, Corollaries 6.2.1 and 6.2.2) according to whether $Q_{t-1} = 0$ (the kernel of the accumulated data is a null space), or $C_t = 0$ (the arriving data is fully dependent on the accumulated data). In particular, Scenario III links the BRMP to the well-known RLS.

6.2.1 Block-wise Recursion by Decomposing the MP inverse

Following the partitions in (6.5), the estimated weight vector based on the accumulated data block is

$$\mathbf{w}^{(t)} = \mathbf{H}_t^\dagger \mathbf{z}_t. \quad (6.6)$$

At $t = T$, the batch-wise solution $\hat{\mathbf{w}} = \hat{\mathbf{w}}^{(T)}$ is obtained. To achieve this equivalent replacement of (6.3), the key is to establish the contribution of $\mathbf{w}^{(t-1)}$ to $\mathbf{w}^{(t)}$, such that the weight vector can be updated using the historical knowledge in a block-wise recursive way.

To proceed, the block-wise partitioning of the MP inverse—the building block of the proposed BRMP—is revisited in Lemma 6.2.1.

Lemma 6.2.1. (see Theorem 2(a) [66]) *The MP inverse of any matrix $\mathbf{A} = \begin{bmatrix} \mathbf{U} & \mathbf{V} \end{bmatrix}$ can be written in the following equivalent form*

$$\mathbf{A}^\dagger = \begin{bmatrix} \mathbf{U}^\dagger - \mathbf{U}^\dagger \mathbf{V} \mathbf{C}^\dagger - \mathbf{U}^\dagger \mathbf{V} \mathbf{V} & \\ & \mathbf{C}^\dagger + \mathbf{V} \end{bmatrix} \quad (6.7)$$

where

$$\mathbf{C} = (\mathbf{I} - \mathbf{U} \mathbf{U}^\dagger) \mathbf{V} \quad (6.8)$$

$$\mathbf{K} = (\mathbf{I} + (\mathbf{I} - \mathbf{C}^\dagger \mathbf{C}) \mathbf{V}^T \mathbf{U}^{\dagger T} \mathbf{U}^\dagger \mathbf{V} (\mathbf{I} - \mathbf{C}^\dagger \mathbf{C}))^{-1} \quad (6.9)$$

$$\mathbf{V} = (\mathbf{I} - \mathbf{C}^\dagger \mathbf{C}) \mathbf{K} \mathbf{V}^T \mathbf{U}^{\dagger T} \mathbf{U}^\dagger (\mathbf{I} - \mathbf{V} \mathbf{C}^\dagger) \quad (6.10)$$

in which $()^{\dagger T}$ means an MP inverse followed by a matrix transpose.

Lemma 6.2.1 is the generalized version of the sample-wise RMP [41], which establishes the connection between the MP inverses, i.e., $\mathbf{A}^\dagger$ and $\mathbf{U}^\dagger$. This connection is one of the fundamental elements in the proposed method. As the proposed method will be derived based on the row-wise decomposition, a transposed form can be obtained in the following lemma.

Lemma 6.2.2. *The MP inverse of a block-wise partitioned matrix $\mathbf{H}_t = [\mathbf{H}_{t-1}^T \quad \mathbf{H}_t^T]^T$ can be written in the following equivalent form*

$$\mathbf{H}_t^\dagger = \begin{bmatrix} \mathbf{H}_{t-1}^\dagger - \mathbf{C}_t^\dagger \mathbf{H}_t \mathbf{H}_{t-1}^\dagger - \mathbf{V}_t \mathbf{H}_t \mathbf{H}_{t-1}^\dagger & \mathbf{C}_t^\dagger + \mathbf{V}_t \end{bmatrix} \quad (6.11)$$

where

$$\mathbf{C}_t = \mathbf{H}_t(\mathbf{I} - \mathbf{H}_{t-1}^\dagger \mathbf{H}_{t-1}) \quad (6.12)$$

$$\mathbf{K}_t = (\mathbf{I} + (\mathbf{I} - \mathbf{C}_t \mathbf{C}_t^\dagger) \mathbf{H}_t \mathbf{H}_{t-1}^\dagger \mathbf{H}_{t-1}^{\dagger T} \mathbf{H}_t^T (\mathbf{I} - \mathbf{C}_t \mathbf{C}_t^\dagger))^{-1} \quad (6.13)$$

$$\mathbf{V}_t = (\mathbf{I} - \mathbf{C}_t^\dagger \mathbf{H}_t) \mathbf{H}_{t-1}^\dagger \mathbf{H}_{t-1}^{\dagger T} \mathbf{H}_t^T \mathbf{K}_t (\mathbf{I} - \mathbf{C}_t \mathbf{C}_t^\dagger). \quad (6.14)$$

Proof. The proof can be obtained by conducting matrix transposes within equations (6.7)-(6.10) in Lemma 6.2.1. $\square$

Lemma 6.2.2 allows us to avoid re-computing $\mathbf{H}_t^\dagger$ directly from $\mathbf{H}_t$ when $\mathbf{H}_{t-1}^\dagger$ is available. Although inverse-like operations still exist (e.g., $\mathbf{C}_t^\dagger$), the involved matrices now have sizes corresponding to those of $\mathbf{H}_t$ and $\mathbf{H}_{t-1}$. This is meaningful as these sizes are smaller than the size of the original matrix $\mathbf{H}_t$. In this way, the memory consumption can potentially be reduced.

In particular, $\mathbf{C}_t$ is referred to as the *residual matrix* at step t , which isolates the component from $\mathbf{H}_t$ that is linearly independent of the accumulated data. To show this, the following linear matrix equation is examined:

$$\mathbf{W} \mathbf{H}_{t-1} = \mathbf{H}_t$$

where $\mathbf{W}$ is an unknown parameter matrix. The LS solution to the above linear problem is $\hat{\mathbf{W}} = \mathbf{H}_t \mathbf{H}_{t-1}^\dagger$. The residual of such an estimation is

$$\mathbf{H}_t - \hat{\mathbf{W}} \mathbf{H}_{t-1} = \mathbf{H}_t - \mathbf{H}_t \mathbf{H}_{t-1}^\dagger \mathbf{H}_{t-1} = \mathbf{C}_t \quad (6.15)$$

revealing that $\mathbf{C}_t$ is the residual between $\mathbf{H}_t$ and its linear estimation $\mathbf{H}_t \mathbf{H}_{t-1}^\dagger \mathbf{H}_{t-1}$. By subtracting $\mathbf{H}_t$'s linear dependency on $\mathbf{H}_{t-1}$, the residual matrix $\mathbf{C}_t$ thus keeps

6.2. BLOCK-WISE RECURSIVE MOORE-PENROSE INVERSE

the part of $\mathcal{H}_t$ that is linearly independent of $\mathbf{H}_{t-1}$.

Based on Lemma 6.2.2, the MP inverse partitioning technique can be applied to building a block-wise recursive implementation of (6.3). This is summarized in the following theorem.

Theorem 6.2.1. *For any $t \geq 2$, the weight $\hat{\mathbf{w}}^{(t)}$ in (6.6) can be obtained based on $\hat{\mathbf{w}}^{(t-1)}$ through*

$$\hat{\mathbf{w}}^{(t)} = \hat{\mathbf{w}}^{(t-1)} + \mathbf{G}_t(\mathbf{z}_t - \mathcal{H}_t \hat{\mathbf{w}}^{(t-1)}) \quad (6.16)$$

where

$$\begin{aligned} \mathbf{C}_t &= \mathcal{H}_t(\mathbf{I} - \mathbf{H}_{t-1}^\dagger \mathbf{H}_{t-1}) \\ \mathbf{K}_t &= (\mathbf{I} + (\mathbf{I} - \mathbf{C}_t \mathbf{C}_t^\dagger) \mathcal{H}_t \mathbf{H}_{t-1}^\dagger \mathbf{H}_{t-1}^{\dagger T} \mathcal{H}_t^T (\mathbf{I} - \mathbf{C}_t \mathbf{C}_t^\dagger))^{-1} \\ \mathbf{V}_t &= (\mathbf{I} - \mathbf{C}_t^\dagger \mathcal{H}_t) \mathbf{H}_{t-1}^\dagger \mathbf{H}_{t-1}^{\dagger T} \mathcal{H}_t^T \mathbf{K}_t (\mathbf{I} - \mathbf{C}_t \mathbf{C}_t^\dagger) \\ \mathbf{G}_t &= \mathbf{C}_t^\dagger + \mathbf{V}_t \\ \mathbf{H}_t^\dagger &= \begin{bmatrix} \mathbf{H}_{t-1}^\dagger - \mathbf{C}_t^\dagger \mathcal{H}_t \mathbf{H}_{t-1}^\dagger - \mathbf{V}_t \mathcal{H}_t \mathbf{H}_{t-1}^\dagger & \mathbf{C}_t^\dagger + \mathbf{V}_t \end{bmatrix}. \end{aligned}$$

Proof. See the proof in Appendix A.3.1. □

Theorem 6.2.1 establishes a direct connection between the weight $\hat{\mathbf{w}}^{(t)}$ at step t and the previously computed weight $\hat{\mathbf{w}}^{(t-1)}$ at step $t - 1$ in a block-wise recursive way. Capitalizing on the historical $\hat{\mathbf{w}}^{(t-1)}$, it enables the update of $\hat{\mathbf{w}}^{(t)}$ with smaller computing blocks and hence less memory requirement than the bulk matrix formulation in (6.6). Using Theorem 6.2.1, the LS solution in (6.6) or (6.3) can therefore be implemented in a block-wise recursive form initialized by $\hat{\mathbf{w}}^{(1)} = \mathbf{H}_1^\dagger \mathbf{z}_1$ and $\mathbf{H}_1^\dagger = \mathcal{H}_1^\dagger$. Although the calculations for $\mathbf{C}_t$, $\mathbf{K}_t$ and $\mathbf{V}_t$ are not trivial, Theorem 6.2.1 offers a

6.2. BLOCK-WISE RECURSIVE MOORE-PENROSE INVERSE

feasible replacement for the batch-wise LS solution.

In particular, $\mathbf{G}_t$ is termed the *gain* matrix since it scales the error between the arriving label $\mathbf{z}_t$ and the predicted label $\mathcal{H}_t \hat{\mathbf{w}}^{(t-1)}$. According to the update rule, the residual matrix $\mathbf{C}_t$ contributes significantly to the gain matrix in the form of its MP inverse.

Although (6.6) can be recursively implemented as shown in Theorem 6.2.1, an additional memory issue caused by $\mathbf{H}_t$ and $\mathbf{H}_t^\dagger$ needs to be addressed. This is because more storage is needed for $\mathbf{H}_t^\dagger$ than for $\mathbf{H}_{t-1}^\dagger$ after each iteration, which is also true between $\mathbf{H}_t$ and $\mathbf{H}_{t-1}$. However, it is also noticed that these memory-increasing matrices always exist in the form of a product pair (e.g., $\mathbf{H}_t^\dagger \mathbf{H}_t$ and $\mathbf{H}_t^\dagger \mathbf{H}_t^{\dagger T}$). It is discovered that these product pairs can be reformulated into recursive forms (see Lemmas 6.2.3 and 6.2.4) which eliminate the need for extra memory during the iterations.

Lemma 6.2.3. *Suppose matrix $\mathbf{Q}_t \in \mathbb{R}^{K \times K}$ is defined by*

$$\mathbf{Q}_t = \mathbf{I} - \mathbf{H}_t^\dagger \mathbf{H}_t. \quad (6.17)$$

Then

$$\mathbf{Q}_t = \mathbf{Q}_{t-1} - \mathbf{G}_t \mathcal{H}_t \mathbf{Q}_{t-1}. \quad (6.18)$$

Proof. See the proof in Appendix A.3.2. □

$\mathbf{Q}_t$ is termed the *kernel projector* which projects a matrix onto the kernel of $\mathbf{H}_t$. This projector is closely linked to the residual matrix, i.e., $\mathbf{C}_t = \mathcal{H}_t(\mathbf{I} - \mathbf{H}_{t-1}^\dagger \mathbf{H}_{t-1}) = \mathcal{H}_t \mathbf{Q}_{t-1}$, which would mean that the arriving data $\mathcal{H}_t$ is projected onto the kernel of the accumulated data $\mathbf{H}_{t-1}$. In other words, the projector keeps the part of the

6.2. BLOCK-WISE RECURSIVE MOORE-PENROSE INVERSE

arriving block that resides in the null space of the accumulated data. That is, $\mathbf{Q}_{t-1}$ isolates the part of the arriving block that is independent of the accumulated data, which aligns with the previous interpretation of $\mathbf{C}_t$.

Remark 6.2.1. *The kernel projector $\mathbf{Q}_t$ has a size of $K \times K$ determined by the feature size K while being detached from the sample size N . Unlike the feature matrix $\mathbf{H}_t$ with a size that grows along with t , treating $\mathbf{H}_t^\dagger \mathbf{H}_t$ as a whole only requires a fixed memory size proportional to the size of the $K \times K$ kernel projector $\mathbf{Q}_t$. This is how the memory usage can be further reduced during the recursion, which is particularly helpful for $N \gg K$. In a similar manner, $\mathbf{H}_t^\dagger \mathbf{H}_t^{\dagger T}$ can be recursively implemented as shown in the following lemma.*

Lemma 6.2.4. *Suppose*

$$\mathbf{R}_t = \mathbf{H}_t^\dagger \mathbf{H}_t^{\dagger T} \in \mathbb{R}^{K \times K}. \quad (6.19)$$

Then

$$\mathbf{R}_t = (\mathbf{I} - \mathbf{G}_t \mathbf{H}_t) \mathbf{R}_{t-1} (\mathbf{I} - \mathbf{G}_t \mathbf{H}_t)^T + \mathbf{G}_t \mathbf{G}_t^T.$$

Proof. See the proof in Appendix A.3.3. □

Lemmas 6.2.3 and 6.2.4 have now eliminated the full sample evaluation of $\mathbf{H}_t^\dagger$ and $\mathbf{H}_t$, solving the memory-increasing problem during iteration. Interestingly, these matrix manipulations are quite similar to the so-called *kernel trick* [92], which adopts the correlation matrix to avoid direct computations of the intermediate matrices.

Combining Theorem 6.2.1 with Lemmas 6.2.3 and 6.2.4, now the proposed BRMP is ready to be presented in the general scenario in Theorem 6.2.2.

Theorem 6.2.2. *For any $t \geq 1$, the weight $\hat{\mathbf{w}}^{(t)}$ in (6.6) can be obtained based on $\hat{\mathbf{w}}^{(t-1)}$ through:*

$$\hat{\mathbf{w}}^{(t)} = \hat{\mathbf{w}}^{(t-1)} + \mathbf{G}_t(\mathbf{z}_t - \mathbf{H}_t \hat{\mathbf{w}}^{(t-1)}) \quad (6.20)$$

where

$$\mathbf{C}_t = \mathbf{H}_t \mathbf{Q}_{t-1} \quad (6.21)$$

$$\mathbf{K}_t = (\mathbf{I} + (\mathbf{I} - \mathbf{C}_t \mathbf{C}_t^\dagger) \mathbf{H}_t \mathbf{R}_{t-1} \mathbf{H}_t^T (\mathbf{I} - \mathbf{C}_t \mathbf{C}_t^\dagger))^{-1} \quad (6.22)$$

$$\mathbf{V}_t = (\mathbf{I} - \mathbf{C}_t^\dagger \mathbf{H}_t) \mathbf{R}_{t-1} \mathbf{H}_t^T \mathbf{K}_t (\mathbf{I} - \mathbf{C}_t \mathbf{C}_t^\dagger) \quad (6.23)$$

$$\mathbf{G}_t = \mathbf{C}_t^\dagger + \mathbf{V}_t \quad (6.24)$$

$$\mathbf{Q}_t = \mathbf{Q}_{t-1} - \mathbf{G}_t \mathbf{H}_t \mathbf{Q}_{t-1} \quad (6.25)$$

$$\mathbf{R}_t = (\mathbf{I} - \mathbf{G}_t \mathbf{H}_t) \mathbf{R}_{t-1} (\mathbf{I} - \mathbf{G}_t \mathbf{H}_t)^T + \mathbf{G}_t \mathbf{G}_t^T \quad (6.26)$$

with a “zero” initialization

$$\hat{\mathbf{w}}^{(0)} = \mathbf{0}, \mathbf{Q}_0 = \mathbf{I}, \mathbf{R}_0 = \mathbf{0}. \quad (6.27)$$

Proof. See the proof in Appendix A.3.4. □

According to Theorem 6.2.2, with initialization (6.27), one can update the weights by simply fetching a small block of data (i.e., $\mathbf{H}_t$ and $\mathbf{z}_t$) to join the calculation in each iteration. The final updated weight $\hat{\mathbf{w}}^{(T)}$ (after iterating through the entire dataset) will be identical to the batch version in (6.3). Since neither approximation nor assumption is involved during the derivation, the proposed BRMP is an equivalent recursive implementation of the LS solution.

Remark 6.2.2. *Breaking the bulk matrix operations in (6.3) into a block-wise recursive computation in (6.20)-(6.26) introduces several benefits. Firstly, it allows the adjustment of the weight vector $\mathbf{w}$ if an arbitrary number of new samples come in under any circumstances. This is particularly desirable when a system needs to update itself in real time based on the incoming data stream such as real-time object tracking (e.g., face tracking [93]). Secondly, it is observed that the BRMP utilizes significantly smaller matrices (e.g., $\mathbf{H}_t$) than those utilized in (6.3) (e.g., $\mathbf{H}$). More importantly, the BRMP retains a small memory consumption during the iterations. This can be verified by looking at the matrix sizes involved in the computation. All the matrices in the updating steps (e.g., $\mathbf{G}_t$, $\mathbf{V}_t$ and $\mathbf{K}_t$) have the sizes corresponding to either the block size n_t or the feature size K . Since usually both n_t and K are smaller than size N of the dataset, the BRMP could reduce the memory consumption. The reduction will be more substantial if $N \gg \{n_t, K\}$ (see experiments in Table 6.7 for verification).*

On the other hand, unlike the sample-wise recursive methods, the BRMP processes several samples at the same time. This allows the algorithm to benefit from the parallelization capability of the modern computing devices, leading to a significant improvement of efficiency. The demonstration will be given later in the experiments (see Table 6.3).

The proposed BRMP belongs to the category of recursive approaches. However, it is more general than the traditional ones, such as the RLS. This is because these traditional recursive methods are often derived based on the Woodbury matrix identity which requires a full-rank $\mathbf{H}_t^T \mathbf{H}_t$ (e.g., see Lemma 12.2 in [94]). The BRMP is formulated based on the decomposition of the MP inverse, which works for matrices with arbitrary conditions. In the subsequent development, it is shown that the RLS

is actually a special case of the proposed BRMP.

6.2.2 The Switching Structure and Linkage to RLS

The general form of BRMP has been developed in Theorem 6.2.2 in which the residual matrix $\mathbf{C}_t$ plays an essential role in the recursive formulation. However, scenarios with $\mathbf{C}_t = \mathbf{0}$, i.e., an empty kernel of the accumulated data $\mathbf{H}_{t-1}$, can be encountered during the update. These scenarios can potentially improve the efficiency of the proposed method by eliminating the computations involving $\mathbf{C}_t$ or $\mathbf{C}_t^\dagger$.

Specifically, there are two cases where $\mathbf{C}_t$ becomes a zero matrix. These cases are presented in Lemma 6.2.5 and 6.2.6 respectively in the following development.

Lemma 6.2.5. *The residual matrix $\mathbf{C}_t = \mathbf{0}$ if and only if $\mathbf{H}_t$ lies in the row space of the accumulated data $\mathbf{H}_{t-1}$.*

Proof. See the proof in Appendix A.3.5. □

As shown in (6.15), the residual matrix $\mathbf{C}_t$ isolates $\mathbf{H}_t$'s linearly independent component of the accumulated data $\mathbf{H}_{t-1}$. $\mathbf{C}_t = \mathbf{0}$ means that there is no component of $\mathbf{H}_t$ that is linearly independent of the accumulated data $\mathbf{H}_{t-1}$, which aligns with the fact that $\mathbf{H}_t$ lies in the row space of $\mathbf{H}_{t-1}$ as shown in Lemma 6.2.5. More importantly, $\mathbf{C}_t = \mathbf{0}$ allows us to remove the terms related to $\mathbf{C}_t$ or $\mathbf{C}_t^\dagger$ in Theorem 6.2.2. The block-wise recursive method can hence be pruned accordingly. This is summarized in the following corollary.

Corollary 6.2.1. *If the residual matrix $\mathbf{C}_t = \mathbf{0}$, (6.20)-(6.26) in Theorem 6.2.2 can be reduced to*

$$\hat{\mathbf{w}}^{(t)} = \hat{\mathbf{w}}^{(t-1)} + \mathbf{G}_t(\mathbf{z}_t - \mathbf{H}_t \hat{\mathbf{w}}^{(t-1)}) \quad (6.28)$$

where

$$\mathbf{G}_t = \mathbf{R}_{t-1} \mathbf{H}_t^T \mathbf{K}_t \quad (6.29)$$

$$\mathbf{K}_t = (\mathbf{I} + \mathbf{H}_t \mathbf{R}_{t-1} \mathbf{H}_t^T)^{-1} \quad (6.30)$$

$$\mathbf{Q}_t = \mathbf{Q}_{t-1} - \mathbf{G}_t \mathbf{H}_t \mathbf{Q}_{t-1} \quad (6.31)$$

$$\mathbf{R}_t = (\mathbf{I} - \mathbf{G}_t \mathbf{H}_t) \mathbf{R}_{t-1} (\mathbf{I} - \mathbf{G}_t \mathbf{H}_t)^T + \mathbf{G}_t \mathbf{G}_t^T. \quad (6.32)$$

Proof. The proof can be immediately obtained by simply removing related terms multiplied by $\mathbf{C}_t$ or $\mathbf{C}_t^\dagger$ in Theorem 6.2.2. $\square$

As shown in Corollary 6.2.1, if $\mathbf{C}_t = \mathbf{0}$, one can reduce the computation rather significantly compared with the original recipe. Note that after the initialization in (6.27), the first update usually happens in the scenario indicated by Theorem 6.2.2. This is because $\mathbf{C}_1 = \mathbf{H}_1 \mathbf{Q}_0 = \mathbf{H}_1 \neq \mathbf{0}$ for non-zero $\mathbf{H}_1$. In addition, since there is a need to detect $\mathbf{C}_t = \mathbf{0}$, numerically the condition should be altered into $\|\mathbf{C}_t\| < \epsilon$, with ϵ being a threshold taking a very small positive value (e.g., 10^{-8}), to take into account the quantization effect.

The switching activity from (6.20)-(6.26) to (6.28)-(6.32) can somewhat be interpreted in the following way. According to (6.15), $\mathbf{C}_t$ is the component of $\mathbf{H}_t$ that is linearly independent of $\mathbf{H}_{t-1}$. It contributes to the gain matrix by extracting the information of $\mathbf{H}_t$ that is beyond the linear exploration of the accumulated knowledge. $\mathbf{C}_t = \mathbf{0}$ would suggest that $\mathbf{H}_t$ carries no such information any more. It thus

leads to a reduction of gain from (6.24) to (6.29) by removing $\mathbf{C}_t^\dagger$ as well as the parts contributed by the residual matrix in $\mathbf{K}_t$ and $\mathbf{R}_{t-1}$.

Alternatively, $\mathbf{C}_t = \mathbf{0}$ could be directly caused by $\mathbf{Q}_{t-1} = \mathbf{0}$ as shown in (6.21). This is detailed in the following lemma with a further development illustrated in Corollary 6.2.2.

Lemma 6.2.6. *The kernel projector $\mathbf{Q}_{t-1} = \mathbf{0}$ if and only if $\mathbf{H}_{t-1}$ has full column rank.*

Proof. See the proof in Appendix A.3.6. □

$\mathbf{Q}_{t-1} = \mathbf{0}$ directly causes $\mathbf{C}_t = \mathbf{0}$ as shown in (6.21). This implication is straightforward as $\mathbf{H}_{t-1}$ having full column rank suggests that its kernel is empty. Projecting any matrix, including $\mathbf{H}_t$, onto an empty kernel leads to a zero matrix. This naturally echoes with the previous interpretation for $\mathbf{C}_t = \mathbf{0}$. Similarly, in terms of detecting $\mathbf{Q}_{t-1} = \mathbf{0}$, $\|\mathbf{Q}_{t-1}\| < \epsilon$ should be adopted instead to eliminate possible errors due to the quantization effect.

On the other hand, $\mathbf{Q}_{t-1} = \mathbf{0}$ results in higher-level simplification for the original recursive algorithm since the related terms to $\mathbf{C}_t$, $\mathbf{C}_t^\dagger$ or $\mathbf{Q}_t$ can be eliminated.

Corollary 6.2.2. *If $\mathbf{Q}_{t-1} = \mathbf{0}$, (6.20)-(6.26) in Theorem 6.2.2 can be reduced to*

$$\hat{\mathbf{w}}^{(t)} = \hat{\mathbf{w}}^{(t-1)} + \mathbf{R}_t \mathbf{H}_t^T (\mathbf{z}_t - \mathbf{H}_t \hat{\mathbf{w}}^{(t-1)}) \quad (6.33)$$

where

$$\mathbf{R}_t = \mathbf{R}_{t-1} - \mathbf{R}_{t-1} \mathbf{H}_t^T (\mathbf{I} + \mathbf{H}_t \mathbf{R}_{t-1} \mathbf{H}_t^T)^{-1} \mathbf{H}_t \mathbf{R}_{t-1}. \quad (6.34)$$

Proof. See the proof in Appendix A.3.7. □

From Corollary 6.2.2, it is important to understand that, once the BRMP switches to (6.33)-(6.34), the switching becomes *permanent* for the subsequent iterations. This can be verified from $\mathbf{Q}_t = \mathbf{Q}_{t-1} - \mathbf{G}_t \mathbf{H}_t \mathbf{Q}_{t-1} = \mathbf{0}$ if $\mathbf{Q}_{t-1} = \mathbf{0}$, which indicates that $\mathbf{Q}_{t_1} = \mathbf{0}$ leads to $\mathbf{Q}_{t_2} = \mathbf{0}$ for index $t_2 > t_1$. It is also the main difference from Corollary 6.2.1 where $\mathbf{Q}_{t-1} \neq \mathbf{0}$ could cause the BRMP to switch back to the scenario in Theorem 6.2.2. This possible switching-back can be shown by

$$\mathbf{C}_{t+1} = \mathbf{H}_{t+1} \mathbf{Q}_t = \mathbf{H}_{t+1} (\mathbf{I} - \mathbf{G}_t \mathbf{H}_t) \mathbf{Q}_{t-1} \stackrel{?}{=} \mathbf{0}$$

where $\stackrel{?}{=}$ indicates *does not necessarily equal to*.

Summary of the Switching Structure

In summary, the BRMP has been proposed and the scenarios with $\mathbf{C}_t = \mathbf{0}$ or $\mathbf{Q}_{t-1} = \mathbf{0}$ have been discussed. The discussion leads to a switching structure of the BRMP among the following three scenarios:

- **Scenario I:** if $\mathbf{C}_t \neq \mathbf{0}$ and $\mathbf{Q}_{t-1} \neq \mathbf{0}$, (6.20)-(6.26) (see Theorem 6.2.2) are adopted.
- **Scenario II:** if $\mathbf{C}_t = \mathbf{0}$ and $\mathbf{Q}_{t-1} \neq \mathbf{0}$, (6.28)-(6.32) (see Corollary 6.2.1) are adopted.
- **Scenario III:** if $\mathbf{Q}_{t-1} = \mathbf{0}$, (6.33)-(6.34) (see Corollary 6.2.2) are adopted.

It is emphasized that once it switches to Scenario III, the subsequent iterations will remain in this scenario. The BRMP with such a switching structure is illustrated in Algorithm 6.1.

Algorithm 6.1 BRMP

Inputs: dataset $\{h_{i1}, h_{i2}, \dots, h_{iK}, z_i\}_{i=1}^N$, block size n_t at step t such that $\sum_{t=1}^T n_t = N$, threshold ϵ .

```

1 Initialize the recursion with  $\hat{\mathbf{w}}^{(0)} = \mathbf{0}$ ,  $\mathbf{Q}_0 = \mathbf{I}$ ,  $\mathbf{R}_0 = \mathbf{0}$ 
2 Set Q flag  $F_Q = False$ 
3 for  $t \leftarrow 1$  to  $T$  do
4     Fetch a data block  $\mathcal{H}_t$  and a label block  $\mathbf{z}_t$  from the dataset according to the block
      size  $n_t$ 
5     if  $F_Q$  is True then
6         | Scenario III: calculate  $\hat{\mathbf{w}}^{(t)}$  and  $\mathbf{R}_t$  using (6.33)-(6.34)
7     else
8         Calculate  $\mathbf{C}_t$  using (6.21)
9         if  $\|\mathbf{C}_t\| < \epsilon$  then
10            | Scenario II: calculate  $\hat{\mathbf{w}}^{(t)}$ ,  $\mathbf{G}_t$ ,  $\mathbf{K}_t$ ,  $\mathbf{Q}_t$  and  $\mathbf{R}_t$  using (6.28)-(6.32) respec-
              | tively
11            | if  $\|\mathbf{Q}_t\| < \epsilon$  then  $F_Q = True$ ;
12        else
13            | Scenario I: calculate  $\hat{\mathbf{w}}^{(t)}$ ,  $\mathbf{G}_t$ ,  $\mathbf{C}_t$ ,  $\mathbf{V}_t$ ,  $\mathbf{K}_t$ ,  $\mathbf{Q}_t$  and  $\mathbf{R}_t$  using (6.20)-(6.26)
              | respectively
              | if  $\|\mathbf{Q}_t\| < \epsilon$  then  $F_Q = True$ ;
14        end
15    end
16 end
Outputs: the weight vector  $\hat{\mathbf{w}}$  or  $\hat{\mathbf{w}}^{(T)}$ .
    
```

BRMP as a generalized RLS

As illustrated in Corollary 6.2.2, the simplified recursive rule (6.33)-(6.34) is actually the update rule for the well-known RLS. This indicates that the RLS is a special case of the proposed method! Note that the RLS discussed here also has a block-wise version that can process multiple samples at each iteration (e.g., see Section 12.2 [94]).

It is not surprising that the BRMP turns out to be a generalization of the RLS. Theoretically, the RLS is a recursive reformulation of the LS solution $\hat{\mathbf{w}}^{(t)} = (\mathbf{H}_t^T \mathbf{H}_t)^{-1} \mathbf{H}_t^T \mathbf{z}_t$, conditioned on a full-rank $\mathbf{H}_t^T \mathbf{H}_t$. The RLS algorithm would fail to recur if such full-rank condition is violated. On the other hand, the proposed BRMP imposes no conditions on the data matrix $\mathbf{H}_t$. A rank-deficient $\mathbf{H}_t$ is allowed during the iterations.

In comparison, the initialization for the RLS is less flexible. The recursion from the BRMP can start at $t = 0$ while the RLS does not permit the “zero” initialization and it initializes the recursion by

$$\hat{\mathbf{w}}^{(1)} = (\mathbf{H}_1^T \mathbf{H}_1)^{-1} \mathbf{H}_1^T \mathbf{z}_1, \quad \mathbf{R}_1 = (\mathbf{H}_1^T \mathbf{H}_1)^{-1}$$

assuming that the first block $\mathbf{H}_1 \in \mathbb{R}^{n_1 \times K}$ has a full column rank. Note that for $\mathbf{H}_1$ to meet such a condition, at least K samples are required for the initialization step. The BRMP, however, allows $\mathbf{H}_1$ to take an arbitrary number of samples.

6.3 Applications to Network Learning

In this section, the proposed BRMP is applied to solving two types of network learning problems, i.e., the learning of a shallow RBF network and a deep MLP, which are good representatives among various neural network variants (e.g., RBF network [22],

MLP [37], fuzzy random neural network [95] and self-organizing neural network [96]). The training of MLP is through reformulating the KARnet proposed in [37].

6.3.1 Block-wise Formulation of the RBF Network

The RBF network has two layers. The first layer transforms the input vector into another feature space using a set of center vectors. Then the second layer maps the transformed features into the output. For an input vector $\mathbf{x}_i$, the output of the RBF network can be written as

$$\phi(\mathbf{x}_i) = \sum_{k=1}^K w_k \rho(\|\mathbf{x}_i - \mathbf{c}_k\|) = \boldsymbol{\rho}_i^T \mathbf{w}$$

where $\mathbf{c}_k$ is the k^{th} center, and

$$\boldsymbol{\rho}_i^T = [\rho(\|\mathbf{x}_i - \mathbf{c}_1\|) \ \rho(\|\mathbf{x}_i - \mathbf{c}_2\|) \ \dots \ \rho(\|\mathbf{x}_i - \mathbf{c}_K\|)]$$

is the mapped feature vector from the input vector $\mathbf{x}_i$ using K center vectors. Here, ρ is a kernel function, usually taking the Gaussian form $\rho(\|\mathbf{x}_i - \mathbf{c}_k\|) = \exp[-\gamma \|\mathbf{x}_i - \mathbf{c}_k\|^2]$ with a width setting parameter γ .

For a set of data samples $\{\mathbf{x}_i, z_i\}_{i=1}^N$, to fit them with an RBF network, it is often formulated as the following minimization problem:

$$\min_{\mathbf{w}} \sum_{i=1}^N (z_i - \phi(\mathbf{x}_i))^2 \tag{6.35}$$

which is a typical LS problem parameterized by $\mathbf{w}^T = [w_1 \ w_2 \ \dots \ w_K]$. The solution

to the minimization problem is given by

$$\hat{\mathbf{w}} = \mathbf{H}^\dagger \mathbf{z} \quad (6.36)$$

where

$$\mathbf{z}^T = [z_1 \ z_2 \ \dots \ z_N]$$

$$\mathbf{H}^T = [\boldsymbol{\rho}_1 \ \boldsymbol{\rho}_2 \ \dots \ \boldsymbol{\rho}_N].$$

It is not hard to see that the solution in (6.36) is almost identical to (6.3) except for an extra feature mapping process in the first layer. Let

$$\mathcal{X}_t^T = [\mathbf{x}_{t_1} \ \mathbf{x}_{t_2} \ \dots \ \mathbf{x}_{t_{n_t}}] \quad (6.37)$$

$$\mathcal{H}_t^T = \rho(\mathcal{X}_t) = [\boldsymbol{\rho}_{t_1} \ \boldsymbol{\rho}_{t_2} \ \dots \ \boldsymbol{\rho}_{t_{n_t}}] \quad (6.38)$$

$$\mathbf{z}_t^T = [z_{t_1} \ z_{t_2} \ \dots \ z_{t_{n_t}}] \quad (6.39)$$

where indexes $t_1, \dots, t_{n_t}$ indicate the data block of n_t samples drawn from the dataset at step t . Then the BRMP can be directly applied to training the RBF network (see Algorithm 6.2). In particular, lines 4 and 5 in Algorithm 6.2 are the only differences from Algorithm 6.1. Note that other similar shallow networks, such as ELM [59], can be trained in a similar manner.

6.3.2 Block-wise Formulation of MLP Learning

The KARnet [38] solves the MLP learning problem by adopting a sequential LS training strategy based on the MP inverse. For an input vector $\mathbf{x}_i$, the output $\hat{\mathbf{y}}_i$ of

6.3. APPLICATIONS TO NETWORK LEARNING

Algorithm 6.2 RBF network trained with BRMP

Inputs: dataset $\{\mathbf{x}_i, z_i\}_{i=1}^N$, block size n_t at step t such that $\sum_{t=1}^T n_t = N$, threshold ϵ .

Initialize the recursion with $\hat{\mathbf{w}}^{(0)} = \mathbf{0}$, $\mathbf{Q}_0 = \mathbf{I}$, $\mathbf{R}_0 = \mathbf{0}$

Set Q flag $F_Q = False$

for $t \leftarrow 1$ **to** T **do**

 Fetch a data block $\mathcal{X}_t$ and a label block $\mathbf{z}_t$ from the dataset according to the block size n_t

 Feed $\mathcal{X}_t$ into the kernel function producing $\mathcal{H}_t$ from (6.38)

 Update $\mathbf{w}^t$ through **steps 5-13** in Algorithm 6.1.

end

Outputs: the weight vector $\hat{\mathbf{w}}$ or $\hat{\mathbf{w}}^{(T)}$.

an L -layer MLP² is

$$\hat{\mathbf{y}}_i^T = f_L(\dots f_2(f_1(\mathbf{x}_i^T \mathbf{W}_1) \mathbf{W}_2) \dots \mathbf{W}_L) \quad (6.40)$$

where $\mathbf{x}_i \in \mathbb{R}^{d_1}$, $\hat{\mathbf{y}}_i \in \mathbb{R}^{d_y}$, and $\mathbf{W}_1 \in \mathbb{R}^{d_1 \times d_2}, \dots, \mathbf{W}_l \in \mathbb{R}^{d_l \times d_{l+1}}, \dots, \mathbf{W}_L \in \mathbb{R}^{d_L \times d_y}$ are the weight matrices for each layer. f_l ($l = 1, \dots, L$) is an activation function in the l^{th} layer which operates in an element-wise manner on the matrices. For a set of data samples $\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^N$, let $\mathbf{X}^T = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N]$, $\hat{\mathbf{Y}}^T = [\hat{\mathbf{y}}_1, \hat{\mathbf{y}}_2, \dots, \hat{\mathbf{y}}_N]$ and $\mathbf{Y}^T = [\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_N]$. The goal of learning is to determine $\mathbf{W}_1, \dots, \mathbf{W}_L$ by minimizing the error between $\mathbf{Y}$ and $\hat{\mathbf{Y}}$.

Let the activation function f_l ($l = 1, \dots, L$) be invertible, i.e., f_l^{-1} exists. Such invertible activation functions are commonly used in multilayer network learning meth-

²The bias is not considered here for simplicity.

6.3. APPLICATIONS TO NETWORK LEARNING

ods [97, 98]. Subsequently, the learning using KARnet [37] can be written as follows:

$$\begin{aligned}
\hat{\mathbf{W}}_1 &= \mathbf{X}^\dagger f_1^{-1}(f_2^{-1}(\dots f_L^{-1}(\mathbf{Y})\mathbf{W}_L^\dagger \dots)\mathbf{W}_2^\dagger) \\
\hat{\mathbf{W}}_2 &= (f_1(\mathbf{X}\hat{\mathbf{W}}_1))^\dagger f_2^{-1}(f_3^{-1}(\dots f_L^{-1}(\mathbf{Y})\mathbf{W}_L^\dagger \dots)\mathbf{W}_3^\dagger) \\
&\vdots \\
\hat{\mathbf{W}}_L &= (f_{L-1}(f_{L-2}(\dots f_1(\mathbf{X}\hat{\mathbf{W}}_1)\dots)\hat{\mathbf{W}}_{L-1}))^\dagger f_L^{-1}(\mathbf{Y})
\end{aligned}$$

which can be simplified as

$$\hat{\mathbf{W}}_l = \mathbf{H}_{l,N}^\dagger \mathbf{Z}_{l,N}, \quad l = 1, \dots, L \quad (6.41)$$

where l, N indicate the l^{th} layer with N samples, and

$$\begin{aligned}
\mathbf{H}_{1,N} &= \mathbf{X}, \quad \mathbf{Z}_{1,N} = f_1^{-1}(\dots f_{L-1}^{-1}(f_L^{-1}(\mathbf{Y})\mathbf{W}_L^\dagger) \dots \mathbf{W}_2^\dagger) \\
\mathbf{H}_{2,N} &= f_1(\mathbf{X}\hat{\mathbf{W}}_1), \quad \mathbf{Z}_{2,N} = f_2^{-1}(\dots f_{L-1}^{-1}(f_L^{-1}(\mathbf{Y})\mathbf{W}_L^\dagger) \dots \mathbf{W}_3^\dagger) \\
&\vdots \\
\mathbf{H}_{L,N} &= f_{L-1}(f_{L-2}(\dots f_1(\mathbf{X}\hat{\mathbf{W}}_1)\dots)\hat{\mathbf{W}}_{L-1}), \quad \mathbf{Z}_{L,N} = f_L^{-1}(\mathbf{Y}).
\end{aligned}$$

The learning starts at $\hat{\mathbf{W}}_1$ after the weight matrices $\mathbf{W}_2, \dots, \mathbf{W}_L$ are randomized.

Since the KARnet adopts the MP inverse, the memory consumption increases for a larger dataset. This disadvantage prohibits the KARnet from training large datasets. Fortunately the main component of the KARNet is the batch-wise LS. Hence, by applying the proposed BRMP, the solution can be calculated in a block-wise manner.

Different from the linear regression problem and the RBF learning, training the KARnet introduces an extra loop due to multiple layers. In each layer, the BRMP can be applied to replace the batch-wise solution given by (6.41). Another difference

6.3. APPLICATIONS TO NETWORK LEARNING

is that the label $\mathbf{y}_i$ of the i^{th} sample is a vector rather than a scalar in the previous cases. Therefore, it is necessary to make changes to the following notations to include the layer index (in layer l):

$$\begin{aligned}\hat{\mathbf{w}}^{(t)} &\rightarrow \hat{\mathbf{W}}_l^{(t)}, \mathbf{V}_t \rightarrow \mathbf{V}_{l,t}, \mathbf{K}_t \rightarrow \mathbf{K}_{l,t}, \mathbf{C}_t \rightarrow \mathbf{C}_{l,t} \\ \mathbf{Q}_t &\rightarrow \mathbf{Q}_{l,t}, \mathbf{R}_t \rightarrow \mathbf{R}_{l,t}, \mathbf{H}_t \rightarrow \mathbf{H}_{l,t}, \mathbf{z}_t \rightarrow \mathbf{Z}_{l,t}, n_t \rightarrow n_{l,t}.\end{aligned}$$

On the other hand, let

$$\begin{aligned}\mathcal{X}_t^T &= [\mathbf{x}_{t_1} \ \mathbf{x}_{t_2} \ \dots \ \mathbf{x}_{t_{n_{l,t}}}] \\ \mathcal{Y}_t^T &= [\mathbf{y}_{t_1} \ \mathbf{y}_{t_2} \ \dots \ \mathbf{y}_{t_{n_{l,t}}}] \end{aligned}$$

be a block of $n_{l,t}$ samples drawn from a dataset. Then, $\mathbf{H}_{l,t}$ and $\mathbf{Z}_{l,t}$ can be built through:

$$\mathbf{H}_{l,t} = f_{l-1}(f_{l-2}(\dots f_1(\mathcal{X}_t \hat{\mathbf{W}}_1) \dots) \hat{\mathbf{W}}_{l-1}) \quad (6.42)$$

$$\mathbf{Z}_{l,t} = f_l^{-1}(\dots f_{L-1}^{-1}(f_L^{-1}(\mathcal{Y}_t) \mathbf{W}_L^\dagger) \dots \mathbf{W}_2^\dagger). \quad (6.43)$$

Accordingly, the weight matrix $\hat{\mathbf{W}}_l$ can now be updated through BRMP by switching among (6.20)-(6.26), (6.28)-(6.32) and (6.33)-(6.34) according to whether $\mathbf{Q}_{l,t} = \mathbf{0}$ or $\mathbf{C}_{l,t} = \mathbf{0}$. These steps are summarized in Algorithm 6.3. For convenience, the method described in the algorithm is termed BRMP-KAR. Note that the reformulation of KARnet is one of the multilayer analytic learning examples. Other MLP analytic learning methods (e.g., CPNet proposed in Chapter 5) can be reformulated in a similar way.

In particular, for the extreme case with block size of 1, the BRMP-KAR of Scenario

Algorithm 6.3 BRMP-KAR

Inputs: dataset $\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^N$, block size $n_{l,t}$ at step t in layer l such that $\sum_{t=1}^T n_{l,t} = N$, threshold ϵ , layer length L .

for $l \leftarrow 1$ **to** L **do**

 Initialize the recursion with $\hat{\mathbf{W}}_l^{(0)} = \mathbf{0}$, $\mathbf{Q}_{l,0} = \mathbf{I}$, $\mathbf{R}_{l,0} = \mathbf{0}$

 Set Q flag $F_Q = False$

for $t \leftarrow 1$ **to** T **do**

 Fetch raw input-output block $\mathbf{x}_t$ and $\mathbf{y}_t$ from the dataset according to the block size $n_{l,t}$

 Form input-output $\mathbf{H}_{l,t}$ and $\mathbf{Z}_{l,t}$ using (6.42) and (6.43)

 Update $\mathbf{W}_l^t$ through **steps 5-13** in Algorithm 6.1.

end

end

Outputs: $\hat{\mathbf{W}}_1, \dots, \hat{\mathbf{W}}_L$.

I has the same formulation as that of the LM-KARnet proposed in [19]. This indicates that the BRMP-KAR (block-wise) can be treated as an extension of the LM-KARnet (sample-wise). Note that the extension is not trivial as the block-wise RMP (see Lemma 6.2.1) is much more sophisticated than the sample-wise RMP (see [41]).

As shown in Algorithm 6.3 and the corresponding theoretical findings in Section 6.2, the BRMP-KAR is a batch-wise recursive formulation of the KARnet as they deliver identical results. Specifically, the BRMP-KAR belongs to the category of *exact recursive learning* which gives an identical solution to its batch-wise counterpart (i.e., KARnet) regardless of the feature matrix's data distribution. Prior works in this category can be found in training linear regression problems [39], and shallow neural network learning [40]. BRMP-KAR may be treated as the multilayer extension of exact recursive learning. Note that this is different from other recursive network learning approaches (e.g., [99]) based on RLS that demand a full-rank feature matrix.

The KARnet learns its parameters in a layer-wise fashion. This allows a reduction of memory since the activation is released for each layer rather than being kept like the

BP-based training paradigm. This property is not only preserved when the MLP is trained by BRMP-KAR but also enhanced. It uncovers the potential of BRMP-KAR to train large networks similar to the SGD-BP based techniques.

The BRMP can also be applied to reformulating the CPNet proposed in Chapter 5. This is because that the CPNet consists of several 2-layer stacking structure. Each of these 2-layer subnetworks can be treated as shallow analytic learning, and reformulation similar to that of RBF network in Section 6.3.1 can be adopted to perform recursive implementation of CPNet, thereby reducing the memory. In comparison, training CPNet directly will require a large amount of memory but with a very high speed, while reformulating it with BRMP trades the speed for memory reduction.

6.3.3 Extension to Networks with Sparse Weights

Consider a sparse network learning problem with $\mathbf{H} \in \mathbb{R}^{N \times K}$, $\mathbf{Z} \in \mathbb{R}^{N \times I}$, and $\mathbf{W} \in \mathbb{R}^{K \times I}$:

$$\begin{aligned} \underset{\mathbf{W}}{\operatorname{argmin}} \quad & \|\mathbf{Z} - \mathbf{H}\mathbf{W}\| \\ \text{subject to} \quad & \text{sparse } \mathbf{W} \end{aligned} \tag{6.44}$$

where the weight matrix $\mathbf{W}$ has sparse elements in each column. That is, $\mathbf{W} = [\mathbf{w}_1 \ \mathbf{w}_2 \ \dots \ \mathbf{w}_I]$ where each $\mathbf{w}_i$ ($i = 1, \dots, I$) is a sparse column vector.

Prior to further development, a *condensed transform* $\mathcal{D}$ and the corresponding *sparse transform* $\mathcal{S}$ are defined as follows. The condensed transform converts a sparse vector into a dense one. For instance, if $\boldsymbol{\alpha} = [0 \ 0 \ 1 \ 0 \ 2 \ 0 \ 0 \ \dots \ 0]^T$ then we have $\boldsymbol{\alpha}^d = \mathcal{D}(\boldsymbol{\alpha}) = [1 \ 2]^T$. The sparse transform inverses such a process. For instance, the previous example can be inverted such that $\boldsymbol{\alpha} = \mathcal{S}(\boldsymbol{\alpha}^d) = [0 \ 0 \ 1 \ 0 \ 2 \ 0 \ 0 \ \dots \ 0]^T$.

Accordingly, let $\mathbf{w}_i^d = \mathcal{D}(\mathbf{w}_i)$, and let $\mathbf{H}_i$ be the corresponding columns from $\mathbf{H}$ according to the locations of the non-zero elements in $\mathbf{w}_i$. For instance, if the 3rd and the 5th elements of $\mathbf{w}_i$ are non-zero, then $\mathbf{H}_i \in \mathbb{R}^{N \times 2}$ is constructed based on the 3rd and the 5th columns of $\mathbf{H}$. Finally, let $\mathbf{z}_i$ be the i^{th} column of $\mathbf{Z}$. The sparse optimization problem in (6.44) is then equivalently formulated as the following unconstrained form

$$\underset{\mathbf{w}_i^d}{\operatorname{argmin}} \quad \left\| \mathbf{z}_i - \mathbf{H}_i \mathbf{w}_i^d \right\|, \quad i = 1, \dots, I \quad (6.45)$$

which leads to the optimal estimate of $\mathbf{w}_i^d$ as follows:

$$\hat{\mathbf{w}}_i^d = \mathbf{H}_i^\dagger \mathbf{z}_i, \quad i = 1, \dots, I. \quad (6.46)$$

Subsequently, the sparse transform $\mathcal{S}$ is applied to each obtained dense vectors, such that the optimal solution to the optimization with sparse weights is $\hat{\mathbf{W}} = [\mathcal{S}(\hat{\mathbf{w}}_1^d) \mathcal{S}(\hat{\mathbf{w}}_2^d) \dots \mathcal{S}(\hat{\mathbf{w}}_I^d)]$.

Here a toy example is provided to demonstrate an extension of the LS solution to sparse-weight network learning. Let $\mathbf{H} = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 6 & 7 & 8 & 9 \end{bmatrix}$, $\mathbf{Z} = \begin{bmatrix} 1 & 2 & 3 \\ 5 & 6 & 7 \end{bmatrix}$. The problem is to solve (6.44) with weight matrix $\mathbf{W} = \begin{bmatrix} w_{11} & 0 & 0 \\ w_{21} & w_{22} & 0 \\ 0 & w_{32} & w_{33} \\ 0 & 0 & w_{43} \end{bmatrix}$, which is of sparse and circulant-like form similar to a CNN weight matrix (See [100] and section 2.4 [101]). Let $\mathbf{z}_1 = [1 \ 5]^T$, $\mathbf{z}_2 = [2 \ 6]^T$, $\mathbf{z}_3 = [3 \ 7]^T$, and $\mathbf{w}_1^d = [w_{11} \ w_{21}]^T$,

$$\mathbf{w}_2^d = [w_{22} \ w_{32}]^T, \mathbf{w}_3^d = [w_{33} \ w_{43}]^T, \text{ which correspond to } \mathbf{H}_1 = \begin{bmatrix} 1 & 2 \\ 6 & 7 \end{bmatrix}, \mathbf{H}_2 = \begin{bmatrix} 2 & 3 \\ 7 & 8 \end{bmatrix},$$

$$\mathbf{H}_3 = \begin{bmatrix} 3 & 4 \\ 8 & 9 \end{bmatrix}. \text{ According to (6.46), the dense weights are estimated as}$$

$$\hat{\mathbf{w}}_1^d = \begin{bmatrix} 0.6 \\ 0.2 \end{bmatrix}, \hat{\mathbf{w}}_2^d = \begin{bmatrix} 0.4 \\ 0.4 \end{bmatrix}, \hat{\mathbf{w}}_3^d = \begin{bmatrix} 0.2 \\ 0.6 \end{bmatrix}.$$

With sparse transform, the weight matrix $\mathbf{W}$ is eventually estimated as

$$\hat{\mathbf{W}} = [\mathcal{S}(\hat{\mathbf{w}}_1^d) \ \mathcal{S}(\hat{\mathbf{w}}_2^d) \ \mathcal{S}(\hat{\mathbf{w}}_3^d)] = \begin{bmatrix} 0.6 & 0 & 0 \\ 0.2 & 0.4 & 0 \\ 0 & 0.4 & 0.2 \\ 0 & 0 & 0.6 \end{bmatrix}$$

which is the optimal solution that can be verified by $\|\mathbf{Z} - \mathbf{H}\hat{\mathbf{W}}\| = 0$.

Since the solution (6.46) to the sparse-weight optimization is identical to (6.3), it can therefore be recursively implemented using BRMP by switching among (6.20)-(6.26), (6.28)-(6.32) and (6.33)-(6.34). That is, the proposed method can be directly applied to network learning with sparse weights.

The BRMP is an exact recursive reformulation of the batch-wise MP-inverse solution. Networks that can be formulated using such solution can directly adopt the BRMP for memory reduction. To the best of our knowledge, the MP-inverse based solution has yet been seen in networks such as CNNs and RNNs that are more sophisticated than the MLPs. Note that related methods that adopted MP inverse in CNNs, such as [21, 102], only focus on addressing the learning of fully-connected parts (i.e.,

MLP) of the networks. The ability of the BRMP to train networks with sparse weights is a good proof of concept for possible extension to more complex networks. Taking the CNN as an example, the convolution layers are essentially MLP layers with sparse circulant weights (see Section 3.1 [103] and Section 9.1 in [83]). Therefore, it is possible to adopt the BRMP to address the memory issue once the CNN weights can be obtained through the MP-inverse solution form. On the other hand, the CNN's circulant weight matrix might have other constraints such as shared elements, which needs further exploration apart from the above condensed-sparse transformation. Hence, this chapter focuses on the applications of the proposed method on training the RBF network and the MLP. It would be the immediate future work to thoroughly investigate into BRMP's extension to more complex networks such as the CNNs and the RNNs.

6.4 Experiments

The proposed BRMP has been theoretically shown to be an equivalent alternative to (6.3). In this section, several experiments are conducted to observe the quantization effect upon the BRMP implementation based on a discrete computing system. The BRMP are mainly compared with its batch-wise counterpart since the goal is to reproduce those results obtained by the batch-wise method. All experiments are run on a workstation with Intel E3-1226 v3 CPU and 8 GB RAM. The validation includes

- Experiments for solving regression problems, aiming to validate the claim of equivalence between the proposed BRMP and the LS solution (6.3). Various properties of the proposed BRMP, such as learning efficiency and memory consumption, are also demonstrated.

6.4. EXPERIMENTS

Table 6.2: Overall comparison between the batch-wise method and the BRMP for $K = 700$, block size 700 and $N = 10000$ by 5 independent runs given randomly generated data.

	batch-wise LS			BRMP				
	T_{LS} (s)	$\ \mathbf{H}\hat{\mathbf{w}} - \mathbf{z}\ $	Max. Mem. (MB)	T_B (s)	$\ \hat{\mathbf{w}}_B - \hat{\mathbf{w}}\ $	$\ \mathbf{H}\hat{\mathbf{w}}_B - \mathbf{z}\ $	Δr_B	Max. Mem. (MB)
1	1.05739	97.38168	257.43	1.91190	0.051	97.38168	-5.2e-07	50.246
2	1.13634	96.17681	257.44	1.64805	1.4e-07	96.17681	-9.2e-12	50.139
3	1.09937	97.19142	259.49	2.04182	0.0036	97.19142	-1.6e-07	49.431
4	1.06538	96.70905	257.51	1.61607	4.2e-08	96.70905	-3.4e-13	51.241
5	1.11136	95.38276	257.43	1.92089	0.00057	95.38276	-1.2e-08	50.864

T_{LS} : Time (seconds) spent using batch-wise LS. Δr_B : Residual error between the batch-wise LS and BRMP (see definition in (6.47)).
 T_B : Time (seconds) spent using BRMP.

- Experiments for solving classification problems, including the training of shallow networks (RBF networks) as well as deep networks (MLPs), in order to evaluate the impact of quantization on BRMP in more complex network learning.

6.4.1 Solving Regression Problems

To validate the equivalence between the BRMP and the batch-wise LS, simulation examples are conducted in the form of (6.1) with various variables. The validation starts with an example to exhibit the basic numerical behaviors of the proposed method. A data matrix $\mathbf{H}$ and a label vector $\mathbf{z}$ with $K = 700$ and $N = 10000$ are generated using a random generator. Subsequently, the regression problem is solved separately using the batch-wise LS and the BRMP. Their performances are shown in Table 6.2. For evaluation, the *residual error* Δr_B is defined using

$$\Delta r_B = \|\mathbf{H}\hat{\mathbf{w}} - \mathbf{z}\| - \|\mathbf{H}\hat{\mathbf{w}}_B - \mathbf{z}\| \quad (6.47)$$

where $\hat{\mathbf{w}}$ and $\hat{\mathbf{w}}_B$ are the weights trained with batch-wise LS and BRMP respectively. The residual error is to show the accuracy performance of the proposed method compared with its LS counterpart. A small residual error would suggest a high resem-

6.4. EXPERIMENTS

Table 6.3: Residual error and time consumption at various block sizes (from 1 to 3000).

block size	1	100	300	500	700	1000	1500	2000	3000
T_B	167.68s	2.68346	1.62506	1.62506	2.04582	2.14676	2.14476	2.99727	3.95072
Δr_B	-1.8e-13	-8.5e-08	1.4e-14	1.1e-13	-1.1e-06	-3.4e-13	1.4e-13	1.8e-13	-2.8e-14

Table 6.4: Residual error under rank-deficient data matrix (rank from 20 to 600).

rank	20	60	100	200	300	400	500	600
Δr_B	0.001	0.0	0.0	-2.8e-14	-1.4e-14	1.4e-14	0.0	2.8e-14

blance to the batch-wise method. In particular, it is possible to obtain a positive residual error, i.e., obtaining a better result than the batch-wise LS, as the quantization effect applies to the batch-wise LS as well.

In Table 6.2, despite the quantization effect, the BRMP is able to almost reproduce (with small $\|\hat{\mathbf{w}}_B - \hat{\mathbf{w}}\|$ and Δr_B) the batch-wise computed result. Although the BRMP is slower than the batch-wise LS, it is an acceptable trade-off between the speed (less than $1\times$ slower) and the memory consumption (reduce by more than 80%).

Subsequently, more evaluations are conducted in terms of several variables, including block size, matrix rank, sparse rate and dataset size.

i) Various block sizes, matrix ranks and sparse rates: Table 6.3 shows the results at various block sizes. For a feature size of 700, the BRMP obtains the highest efficiency at a block size of around 500. As the block size increases, the BRMP still produces close-enough results to those calculated by batch-wise LS. In particular, the extreme case with a block size of 1, i.e., the sample-wise recursive case, is included. As observed in the table, the sample-wise approach spends more than $100\times$ in terms of time consumption in exchange for memory reduction. The proposed BRMP achieves a much better speed-memory trade-off due to tunable block size.

6.4. EXPERIMENTS

Table 6.5: Residual error at various sparse rates (from 20% to 99.99%).

sparse rate	99.99%	99.9%	99%	90%	80%	60%	40%	20%
Δr_B	0.0	-2.8e-14	-1.7e-05	-1.6e-11	-2.1e-11	-2.4e-13	1.4e-14	1.4e-14

Table 6.6: Residual error and memory consumption at various dataset sizes (from 20k to 400k, where “1k” denotes 1000).

dataset size	20k	30k	50k	70k	100k	200k	400k
Δr_B	3.7e-13	-2.8e-14	-1.6e-09	-1.1e-09	-2.4e-08	2.3e-13	-
LS : Max. Mem. (MB)	488.71	690.16	1138.7	1547.7	2258.9	4490.4	OOM
BRMP : Max. Mem. (MB)	49.308	49.385	51.048	50.729	51.048	49.734	54.997

Table 6.4 shows the changes of residual errors under various rank-deficient data matrices. The BRMP shows nearly no gap between the recursive method and the batch-wise LS. This is expected as the BRMP covers the rank-deficient scenario by exploiting the MP inverse decomposition.

The results at various sparse rates³ in Table 6.5. The BRMP is shown to handle the sparse features very well. This is particularly important for training neural networks where a sparse data matrix could be generated after a ReLU [104] activation.

ii) Various dataset size: Table 6.6 shows the residual error and the memory usage for the proposed BRMP. As shown in the table, more training samples barely hurt the recursive method in terms of accuracy. In view of memory usage, the batch-wise LS shows an increasing memory consumption trend towards larger datasets. However, the memory usage can be substantially reduced after applying the BRMP. This leads to a constantly small amount of memory consumption regardless of the size of the dataset. The memory reduction becomes more significant for larger datasets. For instance, for a dataset of 200k samples, the memory requirement can be reduced by 99% for using the BRMP! In particular, when the batch-wise LS encounters OOM at

³The sparse rate is represented by a percentage (see Table 6.5). For instance, 20% indicates 20% of the elements in matrix $\mathbf{H}$ are zeros.

6.4. EXPERIMENTS

Table 6.7: Time and memory usage w.r.t. the feature-sample ratio.

R	10	5	2.5	1	0.2	0.1	0.05	0.025	0.01
T_{LS} (s)	0.0040	0.0110	0.033	0.1750	0.5610	1.7870	1.5710	3.0750	5.9000
T_B (s)	0.1180	0.1210	0.163	0.4090	1.1094	1.5620	2.2260	3.5060	7.3729
LS : Max. Mem. (MB)	0.9257	5.6484	10.502	25.997	87.368	164.66	331.32	635.80	1561.7
BRMP : Max. Mem. (MB)	28.099	29.856	34.886	50.545	50.790	50.033	48.873	51.143	50.127

400k samples, the BRMP can still function properly.

Additionally, a guidance is provided regarding datasets that can benefit from the proposed method. To this end, the time and memory consumption are discussed w.r.t. the feature-sample ratio defined by $R = \frac{\# \text{ of training features}}{\# \text{ of training samples}} = \frac{K}{N}$. The results are shown in Table 6.7 with feature size fixed at 700 and sample size set at (feature size/ R). Naturally, both batch-wise LS and BRMP consume more time with smaller R as the feature matrix $\mathbf{H}$ grows. On the other hand, for small ratios (e.g., $0 < R \leq 0.2$), the BRMP makes a good time-memory trade-off as the memory stays small at the expense of limited extra time compared with that used by the batch-wise LS. That is, when the sample size is considerably larger than the feature size (true in most datasets), the proposed method is an effective memory reducer. This observation verifies the analysis in Remark 6.2.2. Conversely, when the data has more features than the number of samples (i.e., $R > 1$), directly adopting the batch-wise MP inverse is good enough.

In summary, despite of the quantization issue in implementation, the proposed BRMP can numerically reproduce results of the batch-wise LS implementation as evidenced by Tables 6.3-6.5. More importantly, the BRMP could reduce the memory usage (see Table 6.6) to a great extend while preserving the learning efficiency (see Table 6.2).

Figure 6.2: Visualization of the decision boundaries of the RBF networks trained via (a) the batch-wise LS and (b) the proposed BRMP. The blue and red dots are the testing samples.

6.4.2 Solving Classification Problems

As shown in the regression experiments, learning by the BRMP leads to little derivation from the batch-wise method caused by the quantization effect. Here classification experiments are conducted to further evaluate this effect by training the RBF networks and the MLPs.

Training RBF Networks

The RBF networks are trained to show the impact of quantization on the decision making by visualizing the decision boundary. To this end, a 2-arm spiral experiment is conducted, with each arm containing 250 samples while being perturbed by random noise (i.e., rotation angle $+ 0.2 \times \text{rand}$). For this binary classification problem, the label is set to $+1$ for one arm and -1 for the other. Half of the total samples are used for training with the other half for testing. A set of 50 centers is randomly selected from the training samples and γ is set to 5. For the BRMP based training, the block

6.4. EXPERIMENTS

Table 6.8: Dataset specifications.

Dataset	# Attributes	# Training samples	# Testing samples	# Classes
Pendigits	16	7494	3498	10
Optdigits	64	3823	1797	10
Letter	16	20000	5000	26
F-MNIST	784	60000	10000	10
CIFAR-100	3072	50000	10000	100

size is set to 2, which is the same as the feature size. The decision boundaries of the RBF networks trained by the batch-wise LS and the BRMP are shown in Figure 6.2 (a)-(b) respectively.

In this spiral example, training the RBF network with BRMP leads to a larger weight discrepancy of $\|\hat{\mathbf{w}}_B - \hat{\mathbf{w}}\| = 1.1591$ than that in the regression problem. However, the BRMP produces a close-enough decision boundary. This gives evidence that the BRMP could work well for training shallow networks.

Training MLPs by KARnet

The MLP learning is evaluated based on various network depths (up to 5 layers) with an equal number of hidden nodes for each hidden layer. The goal is to show that the BRMP-KAR learns equally well as KARnet while reducing the memory demand.

In the experiments, the block size is chosen to be the same as the feature size in each layer. Apart from reporting the testing accuracy, the *number of different classification outputs*, denoted by N_B , is defined between the classification result of BRMP-KAR and that of the batch-wise KARnet. An N_B would mean that N_B samples are assigned to one class by BRMP-KAR but assigned to another class by KARnet. Larger N_B indicates that the classifier learned by the BRMP-KAR has higher dissimilarity to the KARnet. the SGD-BP is also included as an example to show a comparison of learning efficiency between the iterative-search based methods

6.4. EXPERIMENTS

Table 6.9: MLP learning based on BRMP-KAR in comparison with KARnet. The SGD-BP is also included for the comparison of learning efficiency. Note that the hyperparameters are not tuned to maximize the classification performances and the measurements of time and memory are subject to slight fluctuation.

Dataset	Arch	SGD-BP			KARnet			BRMP-KAR		
		Time (s)	Max. Mem. (MB)	Acc.	Time (s)	Max. Mem. (MB)	Acc.	Time (s)	Max. Mem. (MB)	Acc.
Optdigits	-200-	5.99650	3.4079	0.94880	0.09695	31.412	0.94936	0.17190	5.2019	0.94936
Optdigits	-200-200-	7.34600	5.0258	0.94936	0.21188	35.344	0.95938	0.26085	4.9193	0.95938
Optdigits	-200-200-200-	8.44500	5.6402	0.95437	0.31382	33.030	0.94101	0.45274	5.1692	0.94101
Optdigits	-200-200-200-200-	9.84450	6.4635	0.94713	0.45674	33.747	0.94157	0.63064	4.9930	0.94157
Pendigits	-200-	12.8925	5.1814	0.93053	0.16491	54.780	0.92910	0.38279	5.7180	0.92910
Pendigits	-200-200-	24.9355	10.269	0.95883	0.32881	65.937	0.92682	0.51970	5.3617	0.92682
Pendigits	-200-200-200-	35.7290	12.321	0.96541	0.69560	60.690	0.93625	0.74557	3.9363	0.93625
Pendigits	-200-200-200-200-	48.0720	14.787	0.96541	0.65764	63.885	0.92624	1.02941	5.5910	0.92624
Letter	-500-	30.4820	5.8286	0.76920	2.94131	508.39	0.91200	5.71472	114.02	0.91200
Letter	-500-500-	70.2085	11.440	0.84440	6.33436	621.53	0.92940	12.2899	135.84	0.92940
Letter	-500-500-500-	99.3915	13.132	0.90240	9.47556	621.48	0.93220	19.0720	140.64	0.93220
Letter	-500-500-500-500-	131.523	15.131	0.91840	11.5104	621.63	0.93180	17.9167	140.95	0.93180
F-MNIST	-1000-	891.331	16.310	0.83600	18.5092	2500.1	0.83280	31.9108	117.86	0.83280
F-MNIST	-1000-1000-	1318.33	26.518	0.89350	31.2360	2797.3	0.83080	52.1723	138.88	0.83080
F-MNIST	-1000-1000-1000-	1336.67	32.489	0.89300	40.1941	2797.3	0.82860	63.0001	144.62	0.82860
F-MNIST	-1000-1000-1000-1000-	1675.63	37.622	0.88580	47.2051	2797.4	0.82800	77.0311	144.10	0.82800
F-MNIST	-5000-	1615.07	50.205	0.88400	OOM	OOM	OOM	378.525	2829.6	0.83420
F-MNIST	-5000-5000-	7635.43	298.08	0.88850	OOM	OOM	OOM	1046.63	3446.7	0.82800
F-MNIST	-5000-5000-5000-	13579.5	409.25	0.89810	OOM	OOM	OOM	1461.62	3550.5	0.82050
F-MNIST	-5000-5000-5000-5000-	18304.7	521.44	0.90200	OOM	OOM	OOM	1536.27	3555.2	0.80300
CIFAR-100*	-1000-	1419.69	494.45	0.24660	46.2933	5604.1	0.13420	78.6967	1219.6	0.13410
CIFAR-100*	-1000-1000-	1801.99	470.15	0.25890	52.4085	5570.0	0.14130	87.2226	1293.1	0.14120
CIFAR-100*	-1000-1000-1000-	2135.99	484.99	0.25900	58.4837	5569.7	0.14110	96.3733	1171.2	0.14110
CIFAR-100*	-1000-1000-1000-1000-	2540.28	477.77	0.23260	64.4203	5569.9	0.13730	106.729	1217.7	0.13780

N_B : # classification differences between KARnet trained with BRMP and the original KARnet.

SGD-BP : SGD-BP trains the networks using a batch size of 256 with a learning rate of 0.1 for 100 epochs.

Arch: The architecture is shown by “ $d_2 - \dots - d_n -$ ”. For instance, “-20-20-” indicates a 3-layer network with 20 units in each hidden layer.

*: Note that to match the SGD-BP’s performance on CIFAR datasets, the KARnet would need much wider hidden sizes (e.g., > 10000) [38].

6.4. EXPERIMENTS

(e.g., SGD-BP) and the learning based on MP inverse (e.g., KARnet).

Datasets: The experiments are conducted on datasets of various sizes (see Table 6.8). The Optdigits, Pendigits and Letter datasets are from the UCI Machine Learning Repository⁴. The Fashion-MNIST (F-MNIST) [105] is a dataset of Zalando’s article images⁵. The CIFAR-100 dataset [70] contains various tiny colored images of objects⁶.

Table 6.9 shows the performances of the compared methods. It is observed that BRMP-KAR and KARnet have learned nearly-identical decision boundaries as $N_B = 0$ in most experiments. Although the quantization effect does impose a viable shift on the decision boundary (i.e., $N_B > 0$) on the relatively large dataset CIFAR-100, there is nearly no performance deviation between BRMP-KAR and KARnet. This evidences that BRMP-KAR is able to replicate the classification results of the batch-wise learning. In the meantime, by breaking the bulk matrix computation into small blocks, the BRMP-KAR reduces the memory usage significantly. For instance, for training the MLP with “-1000-1000-” on F-MNIST dataset, the BRMP-KAR reduces the memory usage from 2.80 GB to 0.14 GB. In particular, when the KARnet encounters OOM, the BRMP-KAR is still working properly.

On the other hand, the experiments on SGD-BP empirically show several benefits of learning via the MP inverse (e.g., KARnet). Networks trained with SGD-BP is usually much slower due to multiple visits of the dataset. For instance, training a “-1000-” MLP on F-MNIST by SGD-BP spends more than 890 seconds while the KARnet only needs 18.5 seconds. Also, deeper networks trained by the SGD-BP need more memory as the activation of each layer must be stored before the gradients arrive. The KARnet or the BRMP-KAR shows no such extra demand of memory.

⁴<https://archive.ics.uci.edu/ml/datasets.html>

⁵<https://github.com/zalando-research/fashion-mnist>

⁶<https://www.cs.toronto.edu/~kriz/cifar.html>

In summary, the experiments can be concluded as follows:

1. Experiments for solving the regression problems have shown that, i) the proposed BRMP is indeed an equivalent formulation of the batch-wise LS with neither overwhelming memory consumption nor significant compromised efficiency, and ii) the proposed method performs robustly under various matrix variables, e.g., block size, matrix rank, sparse rate and dataset size.
2. Experiments for training RBF networks demonstrate a high resemblance of decision boundaries between the networks learned by the batch-wise method and its recursive implementation. Experiments for MLP learning have revealed that the proposed BRMP is able to reproduce the classification results obtained by the KARnet with much less memory usage on several benchmark datasets.

6.5 Summary

In this chapter, a block-wise recursive MP inverse formulation (BRMP) is proposed, which allows a network to learn its parameters recursively with an arbitrary number of samples at each iteration. The BRMP releases the learning from overwhelming memory usage without significant efficiency compromise. The underlying relationship between the arriving data and the accumulated historical data is also established with three different scenarios. This shows that the RLS is a special case of the BRMP. Subsequently, the BRMP is applied to training the RBF network as well as the MLP. The conducted experiments show that the proposed block-wise recursive scheme achieves almost the same accuracy performance as that of the batch-wise learning in both regression and classification tasks in the presence of quantization errors. The experiments also empirically demonstrate an outstanding memory reduction property

6.5. SUMMARY

(by 99% for a dataset of 200k samples) without significant efficiency loss (increase the running time by $<1\times$).

Chapter 7

Conclusion and Recommendations

In this chapter, the thesis is concluded in Section 7.1 and suggestions for future exploration are given in Section 7.2.

7.1 Conclusion

In this thesis, acceleration techniques for training DNNs are investigated. The investigation starts with a brief introduction in Chapter 1 of the existing acceleration methods, and is narrowed down to focus of this thesis, the areas of decoupled learning and analytic learning, as well as their research gaps. Then the related works of these two research fields are detailedly reviewed in Chapter 2. Next, new methods have been proposed in Chapter 3 (i.e., FDG) and Chapter 4 (i.e., ADL) in decoupled learning, and in Chapter 5 (i.e., CPNet) and Chapter 6 (i.e., BRMP) in analytic learning. The conclusions of these four chapters are detailed as follows.

BP is a commonly algorithm for training deep neural networks. However, it is inefficient, which has been characterized as three lockings. The proposed FDG in Chapter 3 attains a lock-free structure. The proposed FDG contributes as follows:

7.1. CONCLUSION

- The FDG breaks the forward, backward, and update lockings without introducing extra trainable parameters. A gradient shrinking (GS) process is developed to reduce the stale gradient effect caused by utilizing the delayed gradients.
- It has been illustrated that, in the ideal case, the FDG is able to achieve a linear speedup w.r.t. the number of split modules.
- Convergence analysis of the proposed method is provided, which shows that the method can converge to critical points for the non-convex optimization problem.
- Experiments are conducted by training deep CNNs, revealing that the proposed FDG gives comparable or better results compared with other decoupling methods as well as the standard BP in terms of both generalization and acceleration.

Specifically, the GS process in FDG has been shown to enjoy empirical evidence of mitigating the staleness caused by utilizing DGs. However, no theoretical indication has been made. This has motivated the work of ADL in Chapter 4.

In the decoupled learning area, gradient staleness caused by utilizing order gradients could lead to generalization decrease or even divergence. In Chapter 4, the problem of the gradient staleness issue is investigated in greater detail. The ADL is therefore proposed to mitigate the gradient staleness. The proposed ADL has the following contributions:

- Proposal of a new decoupled learning technique by addressing all the three locking problems in BP.
- Incorporation of an mGA technique into the decoupled learning, which has been shown to reduce the gradient staleness, theoretically and empirically.

7.1. CONCLUSION

- Convergence analysis which shows that our method can converge to critical points.
- Experimental validation on CIFAR-10 and ImageNet datasets. Particularly, we show that the ADL in general outperforms the state-of-the-arts especially under the scenario of a large PS, and has the fastest training speed among the compared methods.

In the area of analytic learning, existing attempts require either invertible activation functions or heavy memory consumption for multilayer network learning. To remove these constraints, in Chapter 5 the CPNet is proposed for multilayer analytic network learning. The proposed CPNet contains several 2-layer modules that conduct their own locally supervised learnings. The main contributions of this work include:

- Introduction of an LE technique, which inserts label information into hidden layers, and thus facilitates a locally supervised learning.
- Proposal of a CP process that allows effective feature extraction in each module, and an analytical learning of weights through the LS estimation.
- Design of a corresponding LD technique that allows each module to give its own prediction. This leads to a multi-exit structure for the proposed CPNet, which is the first of its kind in analytic multilayer classification networks.
- Experimental observations of classification performance and learning efficiency of the proposed network on a variety of datasets show competitive results with several state-of-art methods as well as the benchmark BP learning.

It is a common problem that the single-shot style of analytic learning would encounter OOM issue for large datasets. in Chapter 6, the BRMP is proposed to address

7.1. CONCLUSION

the memory issue. The proposed BRMP is an equivalent formulation to the batch-wise LS and is able to update the parameters with an arbitrary number of samples at a time. The BRMP contributes in the following aspects:

- The existing block-wise decomposition of the MP inverse is connected to the LS solution.
- The BRMP is evaluated in terms of correlating the newly arriving data with the historical data. This leads to a switching structure among three different scenarios. In particular, the BRMP can be linked to the well-known RLS, showing that the RLS is a special case of the BRMP.
- The developed BRMP is implemented in learning the RBF network and a kernel-and-range based MLP.
- Experiments covering regression and classification tasks are conducted to show the numerical behaviors of the proposed technique in comparison with its batch-wise counterpart.

The BRMP proposed in Chapter 6, on the other hand, allows analytic learning to avoid OOM issue due to computing the weights in a batch-wise manner while preserving its one-epoch training effectiveness. It has been proved that the BRMP is an equivalent formulation to its batch-wise counterpart since neither approximation nor assumption is made in the derivation process. It has also been found that the RLS is one of the BRMP's special case. In addition, the proposed BRMP has been successfully applied to several cases of analytic learning with consistent empirical results to our theoretical claims.

7.2 Recommendations for Further Research

Here, possible further works worthy of exploration are pointed out as follows.

In the area of decoupled learning, there are two existing gaps that motivate our attention for developing future works.

1. Currently decoupled learning has been restricted to image classification task only (e.g., FDG and ADL developed in this Thesis). Introducing decoupled learning to other tasks is needed. Although decoupled learning seems to accelerate feedforward networks in a universal way, extending it to other tasks is not trivial. Take the object detection task as an example. Networks for object detection usually contain various feedbacks between lower and upper layers (e.g., concatenation of multi-scale features), rendering the partitioning and communication of any known decoupled learning inapplicable. Therefore, more sophisticated partitioning and communication strategies are in demand before the decoupled learning family can move to more complex tasks.
2. The proposed methods, the FDG in Chapter 3 and the ADL in Chapter 4, achieved lock-free structures that allow a near-ideal acceleration performance. However, this is accomplished accompanying an increased memory consumption. Methods like the FR [14] and the DSP [74] avoid the memory issue by the recomputation technique, which increases the overall computation workload, leading to a decreased acceleration. Therefore, future developments are needed to eliminate the memory increase while maintaining the near-ideal acceleration performance.

In the line of analytic learning, there are possible developments for the CPNet and the BRMP, which are detailed as follows.

7.2. RECOMMENDATIONS FOR FURTHER RESEARCH

1. It has been shown that the CPNet in Chapter 5 has achieved the analytic learning of MLPs without being constrained by existing limitations. However, the structure is limited to MLPs. It would be much more appealing if analytic learning can be extended to other more complex network structures, such as CNNs or RNNs.
2. Note that the BRMP in Chapter 6 is limited to the low feature-sample ratio case (i.e., $\mathbf{H}$ is a “thin” matrix). It would be beneficial to explore solutions that allow our method to give equally efficient performance in the case of having a high feature-sample ratio (i.e., $\mathbf{H}$ is a “fat” one).
3. The analytic learning in general adopts LS estimators and hence cannot process loss functions beyond linear ones. It is meaningful to extend the analytic learning to include more loss functions.
4. The analytic learning usually adopts random projection and LS solutions, and therefore needs more parameters than that of BP to achieve satisfactory results. It is meaningful to explore options to reduce the number of parameters.

Author's Publications

The work presented in this thesis has been accompanied by the following publications.

Journal Papers

1. **H. Zhuang**, Y. Wang, Q. Liu and Z. Lin, “Fully Decoupled Neural Network Learning Using Delayed Gradients,” *IEEE Trans. Neural Netw. Learn. Syst.*, doi: 10.1109/TNNLS.2021.3069883.
2. **H. Zhuang**, Z. Lin and K-A Toh, “Blockwise Recursive Moore-Penrose Inverse for Network Learning,” *IEEE Trans. Syst. Man Cybern. Syst.*, doi: 10.1109/TSMC.2021.3064241.
3. **H. Zhuang**, Z. Lin and K-A Toh, “Correlation Projection for Analytic Learning of a Classification Network,” *Neural Process. Lett.*, (accepted).
4. **H. Zhuang**, Z. Lin and K-A Toh, “Training a multilayer network with low-memory kernel-and-range projection,” *J Franklin Inst.*, 357.1 (2020): 522-550.

Conference Papers

1. **H. Zhuang**, Z. Weng, F. Luo, Z. Lin, K-A Toh and H. Li, “Mitigating Gradient Staleness in Decoupled Learning for Classification Tasks,” in *International Conference of Machine Learning 2021*, (accepted).

2. **H. Zhuang**, Z. Lin and K-A Toh, “Training Multilayer Neural Networks Analytically Using Kernel Projection,” in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2021, pp. 1–5.
3. **H. Zhuang**, Z. Lin and K-A Toh, “A Low-Memory Learning Formulation for a Kernel-and-Range Network,” in *International Joint Conference on Neural Networks*, Budapest, Hungary, July 2019, pp. 1–6.

Bibliography

- [1] J. A. Suykens and J. Vandewalle, “Training multilayer perceptron classifiers based on a modified support vector method,” *IEEE transactions on Neural Networks*, vol. 10, no. 4, pp. 907–911, 1999.
- [2] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner *et al.*, “Gradient-based learning applied to document recognition,” vol. 86, no. 11. Taipei, Taiwan, 1998, pp. 2278–2324.
- [3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” *Advances in Neural Information Processing Systems*, vol. 25, pp. 1097–1105, 2012.
- [4] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 779–788.
- [5] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

- [6] S. Zagoruyko and N. Komodakis, “Wide residual networks,” in *Proceedings of the British Machine Vision Conference (BMVC)*. BMVA Press, 2016, pp. 87.1–87.12.
- [7] A. Sergeev and M. Del Balso, “Horovod: fast and easy distributed deep learning in tensorflow,” *arXiv:1802.05799*, 2018.
- [8] J. Dean, G. S. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Z. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, and A. Y. Ng, “Large scale distributed deep networks,” in *Advances in Neural Information Processing Systems*, 2012, p. 1223–1231.
- [9] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv:1409.1556*, 2014.
- [10] O. Yadan, K. Adams, Y. Taigman, and M. Ranzato, “Multi-gpu training of convnets,” *arXiv:1312.5853*, 2013.
- [11] M. Jaderberg, W. Czarnecki, S. Osindero, O. Vinyals, A. Graves, D. Silver, and K. Kavukcuoglu, “Decoupled neural interfaces using synthetic gradients,” in *International Conference on Machine Learning*, 2016.
- [12] D. Balduzzi, H. Vanchinathan, and J. Buhmann, “Kickback cuts backprop’s red-tape: Biologically plausible credit assignment in neural networks,” in *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.
- [13] Z. Huo, B. Gu, H. Huang *et al.*, “Decoupled parallel backpropagation with convergence guarantee,” in *International Conference on Machine Learning*, 2018, pp. 2103–2111.

- [14] Z. Huo, B. Gu, and H. Huang, “Training neural networks using features replay,” in *Advances in Neural Information Processing Systems*, 2018, pp. 6659–6668.
- [15] J. Chen, X. Pan, R. Monga, S. Bengio, and R. Jozefowicz, “Revisiting distributed synchronous SGD,” *arXiv:1604.00981*, 2016.
- [16] R. Yu, A. Li, C.-F. Chen, J.-H. Lai, V. I. Morariu, X. Han, M. Gao, C.-Y. Lin, and L. S. Davis, “Nisp: Pruning networks using neuron importance score propagation,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 9194–9203.
- [17] P. Guo, M. R. Lyu, and N. Mastorakis, “Pseudoinverse learning algorithm for feedforward neural networks,” *Advances in Neural Networks and Applications*, pp. 321–326, 2001.
- [18] K.-A. Toh, “Learning from the kernel and the range space,” in *the Proceedings of the 17th 2018 IEEE Conference on Computer and Information Science (ICIS)*. IEEE, Jun. 2018, pp. 417–422.
- [19] H. Zhuang, Z. Lin, and K.-A. Toh, “Training a multilayer network with low-memory kernel-and-range projection,” *Journal of the Franklin Institute*, vol. 357, no. 1, pp. 522–550, 2020.
- [20] K. He, Y. Peng, S. Liu, and J. Li, “Regularized negative label relaxation least squares regression for face recognition,” *Neural Processing Letters*, pp. 1–19, 2020.
- [21] C.-C. J. Kuo, M. Zhang, S. Li, J. Duan, and Y. Chen, “Interpretable convolutional neural networks via feedforward design,” *Journal of Visual Communication and Image Representation*, vol. 60, pp. 346–359, 2019.

- [22] J. Park and I. W. Sandberg, “Universal approximation using radial-basis-function networks,” *Neural Computation*, vol. 3, pp. 246–257, June 1991.
- [23] S. A. Barton, “A Matrix Method for Optimizing a Neural Network,” *Neural Computation*, vol. 3, no. 3, pp. 450–459, Sep. 1991.
- [24] S. Chen, P. Grant, and C. Cowan, “Orthogonal least-squares algorithm for training multioutput radial basis function networks,” in *IEE Proceedings F (Radar and Signal Processing)*, vol. 139, no. 6. IET, 1992, pp. 378–384.
- [25] M. Dawson, J. Olvera, A. Fung, and M. Manry, “Inversion of surface parameters using fast learning neural networks,” in *[Proceedings] IGARSS’92 International Geoscience and Remote Sensing Symposium*, vol. 2. IEEE, 1992, pp. 910–912.
- [26] S. Jang, G. Tan, K. Toh, and A. B. J. Teoh, “Online heterogeneous face recognition based on total-error-rate minimization,” *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 50, no. 4, pp. 1286–1299, 2020.
- [27] D. Martínez-Rego, O. Fontenla-Romero, and A. Alonso-Betanzos, “Nonlinear single layer neural network training algorithm for incremental, nonstationary and distributed learning scenarios,” *Pattern Recognition*, vol. 45, no. 12, pp. 4536–4546, 2012.
- [28] K.-A. Toh, Z. Lin, L. Sun, and Z. Li, “Stretchy binary classification,” *Neural Networks*, vol. 97, pp. 74–91, 2018.
- [29] W. Schmidt, M. Kraaijveld, and R. Duin, “Feedforward neural networks with random weights,” in *Proceedings., 11th IAPR International Conference on Pattern Recognition. Vol. II. Conference B: Pattern Recognition Methodology and Systems*. IEEE, 1992, pp. 1–4.

- [30] D. P. Mesquita, J. P. P. Gomes, and L. R. Rodrigues, “Artificial neural networks with random weights for incomplete datasets,” *Neural Processing Letters*, vol. 50, no. 3, pp. 2345–2372, 2019.
- [31] A. Nøkland and L. H. Eidnes, “Training neural networks with local error signals,” in *International Conference on Machine Learning*, 2019, pp. 4839–4850.
- [32] D. Xiao, B. Li, and Y. Mao, “A multiple hidden layers extreme learning machine method and its application,” *Mathematical Problems in Engineering*, 2017.
- [33] A. Ben-Israel and T. N. E. Greville, *Generalized inverses: theory and applications*, 2nd ed. New York: Springer, 2003.
- [34] H. Robbins and S. Monro, “A stochastic approximation method,” *The annals of mathematical statistics*, pp. 400–407, 1951.
- [35] G.-J. Wang and C.-C. Chen, “A fast multilayer neural-network training algorithm based on the layer-by-layer optimizing procedures,” *IEEE Transactions on Neural Networks*, vol. 7, no. 3, pp. 768–775, 1996.
- [36] S. A. Barton, “A matrix method for optimizing a neural network,” *Neural Computation*, vol. 3, no. 3, pp. 450–459, 1991.
- [37] K.-A. Toh, “Learning from the kernel and the range space,” in *2018 IEEE/ACIS 17th International Conference on Computer and Information Science (ICIS)*. IEEE, 2018, pp. 1–6.
- [38] —, “Kernel and range approach to analytic network learning,” *International Journal of Networked and Distributed Computing*, vol. 7, no. 1, pp. 20–28, 2018.

- [39] J. Zhou, Y. Zhu, X. R. Li, and Z. You, “Variants of the Greville formula with applications to exact recursive least squares,” *SIAM journal on matrix analysis and applications*, vol. 24, no. 1, pp. 150–164, 2002.
- [40] D. Martínez-Rego, O. Fontenla-Romero, and A. Alonso-Betanzos, “Exact incremental learning for a single non-linear neuron based on taylor expansion and Greville formula,” in *Conference of the Spanish Association for Artificial Intelligence*. Springer, 2013, pp. 149–158.
- [41] T. Greville, “Some applications of the pseudoinverse of a matrix,” *SIAM review*, vol. 2, no. 1, pp. 15–22, 1960.
- [42] Y. Chen, X. Sun, and Y. Jin, “Communication-efficient federated deep learning with layerwise asynchronous model update and temporally weighted aggregation,” *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–10, 2019.
- [43] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” in *International Conference on Machine Learning*, 2015, pp. 448–456.
- [44] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1–9.
- [45] E. Belilovsky, M. Eickenberg, and E. Oyallon, “Greedy layerwise learning can scale to imagenet,” in *International Conference on Machine Learning*, 2019, pp. 583–593.

- [46] ———, “Decoupled greedy learning of cnns,” in *International Conference on Machine Learning*, 2020, pp. 736–745.
- [47] H. Mostafa, V. Ramesh, and G. Cauwenberghs, “Deep supervised learning using local errors,” *Frontiers in neuroscience*, vol. 12, p. 608, 2018.
- [48] M. Malinowski, G. Swirszcz, J. Carreira, and V. Patraucean, “Sideways: Depth-parallel training of video models,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2020, pp. 11 834–11 843.
- [49] T. P. Lillicrap, D. Cownden, D. B. Tweed, and C. J. Akerman, “Random synaptic feedback weights support error backpropagation for deep learning,” *Nature communications*, vol. 7, p. 13276, 2016.
- [50] A. Nøkland, “Direct feedback alignment provides learning in deep neural networks,” in *Advances in Neural Information Processing Systems*, 2016, pp. 1037–1045.
- [51] D. Xu, A. Clappison, C. Seth, and J. Orchard, “Symmetric predictive estimator for biologically plausible neural learning,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 9, pp. 4140–4151, 2018.
- [52] S. Bartunov, A. Santoro, B. Richards, L. Marris, G. E. Hinton, and T. Lillicrap, “Assessing the scalability of biologically-motivated deep learning algorithms and architectures,” in *Advances in Neural Information Processing Systems*, 2018, pp. 9368–9378.
- [53] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, A. Senior, P. Tucker, K. Yang, Q. V. Le *et al.*, “Large scale distributed deep networks,” in *Advances in Neural Information Processing Systems*, 2012, pp. 1223–1231.

- [54] X. Lian, Y. Huang, Y. Li, and J. Liu, “Asynchronous parallel stochastic gradient for nonconvex optimization,” in *Advances in Neural Information Processing Systems*, 2015, pp. 2737–2745.
- [55] S. Zheng, Q. Meng, T. Wang, W. Chen, N. Yu, Z.-M. Ma, and T.-Y. Liu, “Asynchronous stochastic gradient descent with delay compensation,” in *International Conference on Machine Learning*, 2017.
- [56] Y. Huang, Y. Cheng, A. Bapna, O. Firat, D. Chen, M. Chen, H. Lee, J. Ngiam, Q. V. Le, Y. Wu *et al.*, “Gpipe: Efficient training of giant neural networks using pipeline parallelism,” in *Advances in Neural Information Processing Systems*, 2019, pp. 103–112.
- [57] K.-A. Toh, “Deterministic neural classification,” *Neural computation*, vol. 20, no. 6, pp. 1565–1595, 2008.
- [58] C. P. C. Guo, Ping and Y. Sun, “An exact supervised learning for a three-layer supervised neural network,” in *Proceedings of International Conference on Neural Information Processing*. IEEE, 1995.
- [59] Guang-Bin Huang, Qin-Yu Zhu, and Chee-Kheong Siew, “Extreme learning machine: a new learning scheme of feedforward neural networks,” in *2004 IEEE International Joint Conference on Neural Networks*, vol. 2, July 2004.
- [60] L. Li, K. Zhao, R. Sun, J. Gan, G. Yuan, and T. Liu, “Parameter-free extreme learning machine for imbalanced classification,” *Neural Processing Letters*, pp. 1–18, 2020.
- [61] P. Guo, X. L. Zhou, and K. Wang, “PILAE: A Non-gradient Descent Learning Scheme for Deep Feedforward Neural Networks,” *Arxiv:1811.01545*, Nov 2018.

- [62] L. Deng and D. Yu, “Deep convex net: A scalable architecture for speech pattern classification,” in *Twelfth annual conference of the international speech communication association*, 2011.
- [63] D. Yu, S. Wang, and L. Deng, “Sequential labeling using deep-structured conditional random fields,” *IEEE Journal of Selected Topics in Signal Processing*, vol. 4, no. 6, pp. 965–973, 2010.
- [64] B. Hutchinson, L. Deng, and D. Yu, “Tensor deep stacking networks,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 1944–1957, 2012.
- [65] K.-A. Toh, “Kernel and range approach to analytic network learning,” *International Journal of Networked and Distributed Computing*, vol. 7, no. 1, pp. 20–28, 2018.
- [66] R. E. Cline, “Representations for the generalized inverse of a partitioned matrix,” *Journal of the Society for Industrial and Applied Mathematics*, vol. 12, no. 3, pp. 588–600, 1964.
- [67] P. Werbos, “Beyond regression: New tools for prediction and analysis in the behavioral sciences,” *Ph. D. dissertation, Harvard University*, 1974.
- [68] A. Krogh and J. Hertz, “A simple weight decay can improve generalization,” *Advances in Neural Information Processing Systems*, vol. 4, pp. 950–957, 1991.
- [69] L. Bottou, F. E. Curtis, and J. Nocedal, “Optimization methods for large-scale machine learning,” *Siam Review*, vol. 60, no. 2, pp. 223–311, 2018.
- [70] A. Krizhevsky and G. Hinton, “Learning multiple layers of features from tiny images,” *Tech. Rep.*, 2009.

- [71] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, “Imagenet large scale visual recognition challenge,” *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.
- [72] K. He, X. Zhang, S. Ren, and J. Sun, “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification,” in *Proceedings of the IEEE international Conference on Computer Vision*, 2015, pp. 1026–1034.
- [73] Johnson, “cnn-benchmarks,” in <https://github.com/jcjohnson/cnn-benchmarks.git>, 2017.
- [74] A. Xu, Z. Huo, and H. Huang, “On the acceleration of deep learning model parallelism with staleness,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2020, pp. 2088–2097.
- [75] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang, “On large-batch training for deep learning: Generalization gap and sharp minima,” *arXiv:1609.04836*, 2016.
- [76] D. Narayanan, A. Harlap, A. Phanishayee, V. Seshadri, N. R. Devanur, G. R. Ganger, P. B. Gibbons, and M. Zaharia, “PipeDream: generalized pipeline parallelism for DNN training,” in *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, 2019, pp. 1–15.
- [77] J. H. Park, G. Yun, C. M. Yi, N. T. Nguyen, S. Lee, J. Choi, S. H. Noh, and Y.-r. Choi, “HetPipe: Enabling large DNN training on (whimpy) heterogeneous gpu clusters through integration of pipelined model parallelism and data parallelism,” *arXiv:2005.14038*, 2020.

- [78] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, “Pytorch: An imperative style, high-performance deep learning library,” in *Advances in Neural Information Processing Systems*, 2019, pp. 8024–8035.
- [79] J. Hu, L. Shen, and G. Sun, “Squeeze-and-excitation networks,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 7132–7141.
- [80] A. Neelakantan, L. Vilnis, Q. V. Le, I. Sutskever, L. Kaiser, K. Kurach, and J. Martens, “Adding gradient noise improves learning for very deep networks,” *arXiv:1511.06807*, 2015.
- [81] K. Potdar, T. S. Pardawala, and C. D. Pai, “A comparative study of categorical variable encoding techniques for neural network classifiers,” *International Journal of Computer Applications*, vol. 175, no. 4, pp. 7–9, 2017.
- [82] K.-A. Toh, “Analytic network learning,” *arXiv:1811.08227*, 2018.
- [83] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.
- [84] M. Stone, “Cross-validatory choice and assessment of statistical predictions,” *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 36, no. 2, pp. 111–147, 1974.
- [85] M. A. Nielsen, *Neural networks and deep learning*. Determination press San Francisco, CA, 2015, vol. 25.
- [86] X. Wang, T. Zhang, and R. Wang, “Noniterative deep learning: Incorporating restricted boltzmann machine into multilayer random weight neural networks,”

- IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 49, no. 7, pp. 1299–1308, 2019.
- [87] D. F. Specht, “Probabilistic neural networks,” *Neural networks*, vol. 3, no. 1, pp. 109–118, 1990.
- [88] V. Nair and G. E. Hinton, “Rectified linear units improve restricted boltzmann machines,” in *International Conference on Machine Learning*, 2010, pp. 807–814.
- [89] K.-A. Toh, Z. Lin, Z. Li, B. Oh, and L. Sun, “Gradient-free learning based on the kernel and the range space,” *arXiv:1810.11581*, 2018.
- [90] S. L. Campbell and C. D. Meyer, *Generalized inverses of linear transformations*. SIAM, 2009, vol. 56.
- [91] H. Zhuang, Z. Lin, and K.-A. Toh, “A low-memory learning formulation for a kernel-and-range network,” in *2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2019, pp. 1–6.
- [92] B. Scholkopf and A. J. Smola, *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2001.
- [93] V. Srisamosorn, N. Kuwahara, A. Yamashita, T. Ogata, and J. Ota, “Human-tracking system using quadrotors and multiple environmental cameras for face-tracking application,” *International Journal of Advanced Robotic Systems*, vol. 14, no. 5, 2017.
- [94] E. K. Chong and S. H. Zak, *An introduction to optimization*. John Wiley & Sons, 2013, vol. 76.

- [95] G. A. Anastassiou, “Univariate fuzzy-random neural network approximation operators,” *Computers & Mathematics with Applications*, vol. 48, no. 9, pp. 1263–1283, 2004.
- [96] G. A. Carpenter and S. Grossberg, *Pattern recognition by self-organizing neural networks*. MIT Press, 1991.
- [97] W. Zhang, Q. M. J. Wu, Y. Yang, T. Akilan, and H. Zhang, “A width-growth model with subnetwork nodes and refinement structure for representation learning and image classification,” *IEEE Transactions on Industrial Informatics*, pp. 1–1, 2020.
- [98] Y. Yang, Y. Wang, and X. Yuan, “Bidirectional extreme learning machine for regression problem and its learning effectiveness,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 23, no. 9, pp. 1498–1505, 2012.
- [99] N. Liang, G. Huang, P. Saratchandran, and N. Sundararajan, “A fast and accurate online sequential learning algorithm for feedforward networks,” *IEEE Transactions on Neural Networks*, vol. 17, no. 6, pp. 1411–1423, 2006.
- [100] H. Sedghi, V. Gupta, and P. M. Long, “The singular values of convolutional layers,” in *International Conference on Learning Representations*, 2018.
- [101] A. Maier, C. Syben, T. Lasser, and C. Riess, “A gentle introduction to deep learning in medical image processing,” *Zeitschrift für Medizinische Physik*, vol. 29, no. 2, pp. 86–101, 2019.
- [102] Y. Yang, J. Q. M. Wu, X. Feng, and A. Thangarajah, “Recomputation of dense layers for the performance improvement of DCNN,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2019.

- [103] H. H. Aghdam and E. J. Heravi, “Guide to convolutional neural networks,” *New York, NY: Springer*, vol. 10, pp. 978–973, 2017.
- [104] X. Glorot, A. Bordes, and Y. Bengio, “Deep sparse rectifier neural networks,” in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, 2011, pp. 315–323.
- [105] H. Xiao, K. Rasul, and R. Vollgraf, “Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms,” *preprint arXiv:1708.07747*, 2017.

Appendix A

Proofs for the Theorems and Lemmas

Here the appendices for the previous chapters are given. The proofs for the theorems and lemmas in Chapters 3, 4 and 6 are given in Sections A.1, A.2 and A.3 respectively.

A.1 Appendix for Chapter 3

A.1.1 Proof of Theorem 3.3.1

Proof. According to Assumption 1, the following inequality holds:

$$f(\boldsymbol{\theta}^{t+1}) \leq f(\boldsymbol{\theta}^t) + (\bar{\mathbf{g}}_{\boldsymbol{\theta}}^t)^T(\boldsymbol{\theta}^{t+1} - \boldsymbol{\theta}^t) + \frac{L}{2} \left\| \boldsymbol{\theta}^{t+1} - \boldsymbol{\theta}^t \right\|_2^2. \quad (\text{A.1})$$

According to (3.12) and (3.17), (A.1) can be rewritten as

$$\begin{aligned}
f(\boldsymbol{\theta}^{t+1}) &\leq f(\boldsymbol{\theta}^t) - \gamma_t \sum_{k=1}^K \beta^{K-k} (\bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^t)^T \mathbf{g}_{\boldsymbol{\theta}_{q(k)}}^{d_{k,t}} + \frac{L\gamma_t^2}{2} \sum_{k=1}^K \left\| \beta^{K-k} \mathbf{g}_{\boldsymbol{\theta}_{q(k)}}^{d_{k,t}} \right\|_2^2 \\
&\leq f(\boldsymbol{\theta}^t) - \gamma_t \sum_{k=1}^K \beta^{K-k} (\bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^t)^T \mathbf{g}_{\boldsymbol{\theta}_{q(k)}}^{d_{k,t}} + \frac{L\gamma_t^2}{2} \sum_{k=1}^K \beta^{K-k} \left\| \mathbf{g}_{\boldsymbol{\theta}_{q(k)}}^{d_{k,t}} \right\|_2^2 \quad (\text{A.2})
\end{aligned}$$

where the second inequality is due to $0 < \beta \leq 1$. (A.2) establishes a module-wise relationship between the loss functions at $t+1$ and t . To prove the statistical convergence, the goal is to show that the expectation of the summation of the second and the third term is bounded. To this end, (A.2) can be further developed such that

$$\begin{aligned}
f(\boldsymbol{\theta}^{t+1}) &\leq f(\boldsymbol{\theta}^t) - \gamma_t \sum_{k=1}^K \beta^{K-k} (\bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^t)^T \mathbf{g}_{\boldsymbol{\theta}_{q(k)}}^{d_{k,t}} + \frac{L\gamma_t^2}{2} \sum_{k=1}^K \beta^{K-k} \left\| \mathbf{g}_{\boldsymbol{\theta}_{q(k)}}^{d_{k,t}} \right\|_2^2 \\
&= f(\boldsymbol{\theta}^t) - \gamma_t \sum_{k=1}^K \beta^{K-k} (\bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^t)^T (\mathbf{g}_{\boldsymbol{\theta}_{q(k)}}^{d_{k,t}} - \bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^t + \bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^t) \\
&\quad + \frac{L\gamma_t^2}{2} \sum_{k=1}^K \beta^{K-k} \left\| \mathbf{g}_{\boldsymbol{\theta}_{q(k)}}^{d_{k,t}} - \bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^t + \bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^t \right\|_2^2 \\
&= f(\boldsymbol{\theta}^t) - \gamma_t \sum_{k=1}^K \beta^{K-k} \left\| \bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^t \right\|_2^2 - \gamma_t \sum_{k=1}^K \beta^{K-k} (\bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^t)^T (\mathbf{g}_{\boldsymbol{\theta}_{q(k)}}^{d_{k,t}} - \bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^t) \\
&\quad + \frac{L\gamma_t^2}{2} \sum_{k=1}^K \beta^{K-k} \left\| \bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^t \right\|_2^2 + \frac{L\gamma_t^2}{2} \sum_{k=1}^K \beta^{K-k} \left\| \mathbf{g}_{\boldsymbol{\theta}_{q(k)}}^{d_{k,t}} - \bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^t \right\|_2^2 \\
&\quad + L\gamma_t^2 \sum_{k=1}^K \beta^{K-k} (\bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^t)^T (\mathbf{g}_{\boldsymbol{\theta}_{q(k)}}^{d_{k,t}} - \bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^t) \\
&= f(\boldsymbol{\theta}^t) - \left(\gamma_t - \frac{L\gamma_t^2}{2} \right) \sum_{k=1}^K \beta^{K-k} \left\| \bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^t \right\|_2^2 + \tilde{Q}_1 + \tilde{Q}_2. \quad (\text{A.3})
\end{aligned}$$

where

$$\tilde{Q}_1 = \frac{L\gamma_t^2}{2} \sum_{k=1}^K \beta^{K-k} \left\| \mathbf{g}_{\boldsymbol{\theta}_{q(k)}}^{d_{k,t}} - \bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^t \right\|_2^2, \quad \tilde{Q}_2 = (L\gamma_t^2 - \gamma_t) \sum_{k=1}^K \beta^{K-k} (\bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^t)^T (\mathbf{g}_{\boldsymbol{\theta}_{q(k)}}^{d_{k,t}} - \bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^t).$$

The expectation of $\tilde{Q}_1$ is bounded by

$$\begin{aligned} \mathbb{E}_{\mathbf{x}_t} \{\tilde{Q}_1\} &= \frac{L\gamma_t^2}{2} \mathbb{E}_{\mathbf{x}_t} \left\{ \sum_{k=1}^K \beta^{K-k} \left\| \mathbf{g}_{\boldsymbol{\theta}_{q(k)}}^{d_{k,t}} - \bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^t \right\|_2^2 \right\} \\ &= \frac{L\gamma_t^2}{2} \mathbb{E}_{\mathbf{x}_t} \left\{ \sum_{k=1}^K \beta^{K-k} \left\| \mathbf{g}_{\boldsymbol{\theta}_{q(k)}}^{d_{k,t}} - \bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^{d_{k,t}} - \bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^t + \bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^{d_{k,t}} \right\|_2^2 \right\} \\ &\leq L\gamma_t^2 \mathbb{E}_{\mathbf{x}_t} \left\{ \sum_{k=1}^K \beta^{K-k} \left\| \mathbf{g}_{\boldsymbol{\theta}_{q(k)}}^{d_{k,t}} - \bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^{d_{k,t}} \right\|_2^2 \right\} + L\gamma_t^2 \sum_{k=1}^K \beta^{K-k} \left\| \bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^{d_{k,t}} - \bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^t \right\|_2^2 \\ &= L\gamma_t^2 \sum_{k=1}^K \beta^{K-k} \mathbb{E}_{\mathbf{x}_t} \left\{ \left\| \mathbf{g}_{\boldsymbol{\theta}_{q(k)}}^{d_{k,t}} - \bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^{d_{k,t}} \right\|_2^2 \right\} + L\gamma_t^2 \sum_{k=1}^K \beta^{K-k} \left\| \bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^{d_{k,t}} - \bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^t \right\|_2^2 \\ &\leq L\gamma_t^2 \sum_{k=1}^K \beta^{K-k} \mathbb{E}_{\mathbf{x}_t} \left\{ \left\| \mathbf{g}_{\boldsymbol{\theta}_{q(k)}}^{d_{k,t}} \right\|_2^2 \right\} + L\gamma_t^2 \sum_{k=1}^K \beta^{K-k} \left\| \bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^{d_{k,t}} - \bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^t \right\|_2^2 \\ &\leq L\gamma_t^2 A \sum_{k=1}^K \beta^{K-k} + L\gamma_t^2 \sum_{k=1}^K \beta^{K-k} \left\| \bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^{d_{k,t}} - \bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^t \right\|_2^2 = L\gamma_t^2 A \sum_{k=1}^K \beta^{K-k} + L\gamma_t^2 \tilde{P}_1 \end{aligned}$$

where the first inequality follows from $\|\mathbf{x} + \mathbf{y}\|_2^2 \leq 2\|\mathbf{x}\|_2^2 + 2\|\mathbf{y}\|_2^2$. The second inequality is from $\mathbb{E}\{\|\epsilon - \mathbb{E}\{\epsilon\}\|_2^2\} \leq \mathbb{E}\{\|\epsilon\|_2^2\} - \|\mathbb{E}\{\epsilon\}\|_2^2 \leq \mathbb{E}\{\|\epsilon\|_2^2\}$ due to gradient unbiasedness (i.e., $\mathbb{E}_{\mathbf{x}_t} \{\mathbf{g}_{\boldsymbol{\theta}_{q(k)}}^{d_{k,t}}\} = \bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^{d_{k,t}}$). The last inequality follows from Assumption 2, and $\tilde{P}_1$ is

bounded by

$$\begin{aligned}
\tilde{P}_1 &= \sum_{k=1}^K \beta^{K-k} \left\| \bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^{d_{k,t}} - \bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^t \right\|_2^2 \\
&\leq L^2 \sum_{k=1}^K \beta^{K-k} \left\| \boldsymbol{\theta}_{q(k)}^t - \boldsymbol{\theta}_{q(k)}^{d_{k,t}} \right\|_2^2 \\
&= L^2 \sum_{k=1}^K \beta^{K-k} \left\| \sum_{j=\max\{0, d_{k,t}\}}^{t-1} (\boldsymbol{\theta}_{q(k)}^{j+1} - \boldsymbol{\theta}_{q(k)}^j) \right\|_2^2 \\
&\leq L^2 \sum_{k=1}^K \beta^{K-k} \sum_{j=\max\{0, d_{k,t}\}}^{t-1} \left\| \boldsymbol{\theta}_{q(k)}^{j+1} - \boldsymbol{\theta}_{q(k)}^j \right\|_2^2 \\
&= L^2 \sum_{k=1}^K \beta^{K-k} \sum_{j=\max\{0, d_{k,t}\}}^{t-1} \gamma_j^2 \beta^{2(K-k)} \left\| \mathbf{g}_{\boldsymbol{\theta}_{q(k)}}^{d_{k,t}} \right\|_2^2 \\
&\leq L^2 A \sum_{k=1}^K \beta^{3(K-k)} \sum_{j=\max\{0, d_{k,t}\}}^{t-1} \gamma_j^2 \\
&\leq \gamma_t^2 L^2 A \sum_{k=1}^K \beta^{3(K-k)} (t - \max\{0, d_{k,t}\}) \\
&\leq \gamma_t^2 L^2 A \sum_{k=1}^K \beta^{3(K-k)} 2(K-k)
\end{aligned}$$

with the first inequality coming from Assumption 1. On the other hand, The expectation of $\tilde{Q}_2$ is bounded by

$$\begin{aligned}
\mathbb{E}_{\mathbf{x}_t} \{\tilde{Q}_2\} &= -(\gamma_t - L\gamma_t^2) \mathbb{E}_{\mathbf{x}_t} \left\{ \sum_{k=1}^K \beta^{K-k} (\bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^t)^T \left(\mathbf{g}_{\boldsymbol{\theta}_{q(k)}}^{d_{k,t}} - \bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^t \right) \right\} \\
&= -(\gamma_t - L\gamma_t^2) \sum_{k=1}^K \beta^{K-k} (\bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^t)^T \left(\mathbf{g}_{\boldsymbol{\theta}_{q(k)}}^{d_{k,t}} - \bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^t \right) \\
&\leq \frac{\gamma_t - L\gamma_t^2}{2} \sum_{k=1}^K \beta^{K-k} \left\| \bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^t \right\|_2^2 + \frac{\gamma_t - L\gamma_t^2}{2} \tilde{P}_1
\end{aligned}$$

where the second equality follows by the unbiased gradient using SGD, and the inequality comes from $\pm \mathbf{x}^T \mathbf{y} \leq \frac{1}{2} \|\mathbf{x}\|_2^2 + \frac{1}{2} \|\mathbf{y}\|_2^2$.

Taking the expectation of both sides in (A.3) and substituting $\tilde{Q}_1$ and $\tilde{Q}_2$, the inequality is rewritten as

$$\begin{aligned}
\mathbb{E}_{\mathbf{x}_t} \{f(\boldsymbol{\theta}^{t+1})\} &\leq f(\boldsymbol{\theta}^t) - \left(\gamma_t - \frac{L\gamma_t^2}{2}\right) \sum_{k=1}^K \beta^{K-k} \left\| \bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^t \right\|_2^2 + L\gamma_t^2 A \sum_{k=1}^K \beta^{K-k} + L\gamma_t^2 \tilde{P}_1 \\
&\quad + \frac{\gamma_t - L\gamma_t^2}{2} \sum_{k=1}^K \beta^{K-k} \left\| \bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^t \right\|_2^2 + \frac{\gamma_t - L\gamma_t^2}{2} \tilde{P}_1 \\
&\leq f(\boldsymbol{\theta}^t) - \frac{\gamma_t}{2} \sum_{k=1}^K \beta^{K-k} \left\| \bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^t \right\|_2^2 + \frac{\gamma_t + L\gamma_t^2}{2} \gamma_t^2 L^2 A \sum_{k=1}^K \beta^{3(K-k)} 2(K-k) \\
&\quad + L\gamma_t^2 A \sum_{k=1}^K \beta^{K-k} \\
&= f(\boldsymbol{\theta}^t) - \frac{\gamma_t}{2} \sum_{k=1}^K \beta^{K-k} \left\| \bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^t \right\|_2^2 \\
&\quad + \gamma_t^2 \left(LA \sum_{k=1}^K \beta^{K-k} + \frac{\gamma_t + L\gamma_t^2}{2} L^2 A \sum_{k=1}^K \beta^{3(K-k)} 2(K-k) \right) \\
&\leq f(\boldsymbol{\theta}^t) - \frac{\gamma_t}{2} \sum_{k=1}^K \beta^{K-k} \left\| \bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^t \right\|_2^2 \\
&\quad + \gamma_t^2 \left(LA \sum_{k=1}^K \beta^{K-k} + LA \sum_{k=1}^K \beta^{3(K-k)} 2(K-k) \right)
\end{aligned}$$

where the last inequality follows from $L\gamma_t \leq 1$ such that $\frac{\gamma_t + L\gamma_t^2}{2} L = \frac{1}{2} (L\gamma_t + (L\gamma_t)^2) \leq$

1. The proof is now completed. $\square$

A.1.2 Proof of Theorem 3.3.2

Proof. By moving $-\frac{\gamma_t}{2}Z_1$ and $\mathbb{E}_{\mathbf{x}_t}\{f(\boldsymbol{\theta}^{t+1})\} - f(\boldsymbol{\theta}^t)$ to the left and the right side of (3.22) respectively, and multiplying both sides by 2, we have

$$\gamma_t Z_1 \leq 2(f(\boldsymbol{\theta}^t) - \mathbb{E}_{\mathbf{x}_t}\{f(\boldsymbol{\theta}^{t+1})\}) + 2\gamma_t^2 Z_2. \quad (\text{A.4})$$

Since $\beta^{K-k} \geq \beta^K$ (for $k \leq K$ and $|\beta| \leq 1$), we have

$$Z_1 \geq \beta^K \sum_{k=1}^K \|\bar{\mathbf{g}}_{q(k)}^t\|_2^2 = \beta^K \|\bar{\mathbf{g}}_{\boldsymbol{\theta}}^t\|_2^2 \quad (\text{A.5})$$

which leads to

$$\gamma_t \|\bar{\mathbf{g}}_{\boldsymbol{\theta}}^t\|_2^2 \leq \frac{2(f(\boldsymbol{\theta}^t) - \mathbb{E}_{\mathbf{x}_t}\{f(\boldsymbol{\theta}^{t+1})\})}{\beta^K} + \frac{2\gamma_t^2 Z_2}{\beta^K}. \quad (\text{A.6})$$

We take the full expectation of (A.6) and it results in

$$\gamma_t \mathbb{E}\{\|\bar{\mathbf{g}}_{\boldsymbol{\theta}}^t\|_2^2\} \leq \frac{2(\mathbb{E}\{f(\boldsymbol{\theta}^t)\} - \mathbb{E}\{f(\boldsymbol{\theta}^{t+1})\})}{\beta^K} + \frac{2\gamma_t^2 Z_2}{\beta^K}. \quad (\text{A.7})$$

By summing both sides of (A.7) from 0 to $T-1$, and dividing it by $\mathcal{T}_T = \sum_{t=0}^{T-1} \gamma_t$, it becomes

$$\begin{aligned} \frac{1}{\mathcal{T}_T} \sum_{t=0}^{T-1} \gamma_t \mathbb{E}\{\|\bar{\mathbf{g}}_{\boldsymbol{\theta}}^t\|_2^2\} &\leq \frac{2(f(\boldsymbol{\theta}^0) - \mathbb{E}\{f(\boldsymbol{\theta}^T)\})}{\beta^K \mathcal{T}_T} + \frac{2Z_2 \sum_{t=0}^{T-1} \gamma_t^2}{\beta^K \mathcal{T}_T} \\ &\leq \frac{2(f(\boldsymbol{\theta}^0) - f(\boldsymbol{\theta}^*))}{\beta^K \mathcal{T}_T} + \frac{2Z_2 \sum_{t=0}^{T-1} \gamma_t^2}{\beta^K \mathcal{T}_T}. \end{aligned}$$

where the last inequality comes from $f(\boldsymbol{\theta}^*) \leq \mathbb{E}\{f(\boldsymbol{\theta}^T)\}$. □

A.1.3 Proof of Theorem 3.3.3

Proof. The proof is started from (A.7) as the constant learning rate is a special case in Theorem 3.3.2. By setting $\gamma_t = \gamma$, (A.7) is rewritten as

$$\mathbb{E}\{\|\bar{\mathbf{g}}_{\boldsymbol{\theta}}^t\|_2^2\} \leq \frac{2(\mathbb{E}\{f(\boldsymbol{\theta}^t)\} - \mathbb{E}\{f(\boldsymbol{\theta}^{t+1})\})}{\beta^K \gamma} + \frac{2\gamma Z_2}{\beta^K}. \quad (\text{A.8})$$

Summing both sides of (A.8) from $t = 0$ to $t = T - 1$ and dividing them by T , it leads to

$$\frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}\{\|\bar{\mathbf{g}}_{\boldsymbol{\theta}}^t\|_2^2\} \leq \frac{2(f(\boldsymbol{\theta}^0) - \mathbb{E}\{f(\boldsymbol{\theta}^T)\})}{\beta^K \gamma T} + \frac{2\gamma Z_2}{\beta^K} \quad (\text{A.9})$$

$$\leq \frac{2(f(\boldsymbol{\theta}^0) - f(\boldsymbol{\theta}^*))}{\beta^K \gamma T} + \frac{2\gamma Z_2}{\beta^K}. \quad (\text{A.10})$$

Substituting $\gamma = \epsilon \sqrt{\frac{f(\boldsymbol{\theta}^0) - f(\boldsymbol{\theta}^*)}{TZ_2}}$ into (A.9), the RHS becomes

$$\begin{aligned} \frac{2(f(\boldsymbol{\theta}^0) - f(\boldsymbol{\theta}^*)) + 2\gamma^2 TZ_2}{\beta^K \gamma T} &= \frac{(2 + 2\epsilon^2)(f(\boldsymbol{\theta}^0) - f(\boldsymbol{\theta}^*))}{\beta^K T \epsilon \sqrt{(f(\boldsymbol{\theta}^0) - f(\boldsymbol{\theta}^*)) / TZ_2}} \\ &= \frac{(2 + 2\epsilon^2)}{\epsilon \beta^K} \sqrt{\frac{(f(\boldsymbol{\theta}^0) - f(\boldsymbol{\theta}^*)) Z_2}{T}}. \end{aligned}$$

Since the left hand side of (A.9) is the average of $\mathbb{E}\{\|\bar{\mathbf{g}}_{\boldsymbol{\theta}}^t\|_2^2\}$ for $t = 0, 1, \dots, T - 1$, we have

$$\min_{t \in \{0, 1, \dots, T-1\}} \mathbb{E}\{\|\bar{\mathbf{g}}_{\boldsymbol{\theta}}^t\|_2^2\} \leq \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}\{\|\bar{\mathbf{g}}_{\boldsymbol{\theta}}^t\|_2^2\} \leq \frac{(2 + 2\epsilon^2)}{\epsilon \beta^K} \sqrt{\frac{(f(\boldsymbol{\theta}^0) - f(\boldsymbol{\theta}^*)) Z_2}{T}}$$

which completes the proof. $\square$

A.2 Appendix for Chapter 4

A.2.1 Proof of Theorem 4.3.1

Proof. To simplify the notations, let $\mathbf{g}_{\boldsymbol{\theta}_{q(k)}}^{U'_s} = \frac{1}{M} \sum_{j=0}^{M-1} \mathbf{g}_{\boldsymbol{\theta}_{q(k)}}^{U_s+j-2(K-k)}$ and $\bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^{U'_s} = \frac{1}{M} \sum_{j=0}^{M-1} \bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^{U_s+j-2(K-k)}$. According to Assumption 4.3.1, the following inequality holds:

$$\begin{aligned} f(\boldsymbol{\theta}^{U_{s+1}}) &\leq f(\boldsymbol{\theta}^{U_s}) + (\bar{\mathbf{g}}_{\boldsymbol{\theta}^{U_s}})^T (\boldsymbol{\theta}^{U_{s+1}} - \boldsymbol{\theta}^{U_s}) + \frac{L}{2} \|\boldsymbol{\theta}^{U_{s+1}} - \boldsymbol{\theta}^{U_s}\|_2^2 \\ &= f(\boldsymbol{\theta}^{U_s}) - \gamma_s \sum_{k=1}^K (\bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^{U_s})^T \mathbf{g}_{\boldsymbol{\theta}_{q(k)}}^{U'_s} + \frac{L\gamma_s^2}{2} \sum_{k=1}^K \|\mathbf{g}_{\boldsymbol{\theta}_{q(k)}}^{U'_s}\|_2^2 \end{aligned} \quad (\text{A.11})$$

which can be further developed such that

$$\begin{aligned} f(\boldsymbol{\theta}^{U_{s+1}}) &\leq f(\boldsymbol{\theta}^{U_s}) - \gamma_s \sum_{k=1}^K (\bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^{U_s})^T (\mathbf{g}_{\boldsymbol{\theta}_{q(k)}}^{U'_s} - \bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^{U_s} + \bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^{U_s}) \\ &\quad + \frac{L\gamma_s^2}{2} \sum_{k=1}^K \|\mathbf{g}_{\boldsymbol{\theta}_{q(k)}}^{U'_s} - \bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^{U_s} + \bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^{U_s}\|_2^2 \\ &= f(\boldsymbol{\theta}^{U_s}) - \gamma_s \sum_{k=1}^K \|\bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^{U_s}\|_2^2 - \gamma_s \sum_{k=1}^K (\bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^{U_s})^T (\mathbf{g}_{\boldsymbol{\theta}_{q(k)}}^{U'_s} - \bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^{U_s}) + \frac{L\gamma_s^2}{2} \sum_{k=1}^K \|\bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^{U_s}\|_2^2 \\ &\quad + \frac{L\gamma_s^2}{2} \sum_{k=1}^K \|\mathbf{g}_{\boldsymbol{\theta}_{q(k)}}^{U'_s} - \bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^{U_s}\|_2^2 + L\gamma_s^2 \sum_{k=1}^K (\bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^{U_s})^T (\mathbf{g}_{\boldsymbol{\theta}_{q(k)}}^{U'_s} - \bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^{U_s}) \\ &= f(\boldsymbol{\theta}^{U_s}) - \left(\gamma_s - \frac{L\gamma_s^2}{2} \right) \sum_{k=1}^K \|\bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^{U_s}\|_2^2 + \tilde{Q}_1 + \tilde{Q}_2 \end{aligned} \quad (\text{A.12})$$

where

$$\tilde{Q}_1 = \frac{L\gamma_s^2}{2} \sum_{k=1}^K \|\mathbf{g}_{\boldsymbol{\theta}_{q(k)}}^{U'_s} - \bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^{U_s}\|_2^2, \quad \tilde{Q}_2 = (L\gamma_s^2 - \gamma_s) \sum_{k=1}^K (\bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^{U_s})^T (\mathbf{g}_{\boldsymbol{\theta}_{q(k)}}^{U'_s} - \bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^{U_s}).$$

The expectation of $\tilde{Q}_1$ is bounded by

$$\begin{aligned}
\mathbb{E}_{\mathbf{x}}\{\tilde{Q}_1\} &= \frac{L\gamma_s^2}{2} \mathbb{E}_{\mathbf{x}} \left\{ \sum_{k=1}^K \left\| \mathbf{g}_{\boldsymbol{\theta}_{q(k)}}^{U'_s} - \bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^{U_s} \right\|_2^2 \right\} \\
&= \frac{L\gamma_s^2}{2} \mathbb{E}_{\mathbf{x}} \left\{ \sum_{k=1}^K \left\| \mathbf{g}_{\boldsymbol{\theta}_{q(k)}}^{U'_s} - \bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^{U'_s} - \bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^{U_s} + \bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^{U'_s} \right\|_2^2 \right\} \\
&\leq L\gamma_s^2 \mathbb{E}_{\mathbf{x}} \left\{ \sum_{k=1}^K \left\| \mathbf{g}_{\boldsymbol{\theta}_{q(k)}}^{U'_s} - \bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^{U'_s} \right\|_2^2 \right\} + L\gamma_s^2 \sum_{k=1}^K \left\| \bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^{U'_s} - \bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^{U_s} \right\|_2^2 \\
&= L\gamma_s^2 \mathbb{E}_{\mathbf{x}} \left\{ \left\| \mathbf{g}_{\boldsymbol{\theta}}^{U'_s} - \bar{\mathbf{g}}_{\boldsymbol{\theta}}^{U'_s} \right\|_2^2 \right\} + L\gamma_s^2 \sum_{k=1}^K \left\| \bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^{U'_s} - \bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^{U_s} \right\|_2^2 \\
&\leq L\gamma_s^2 \mathbb{E}_{\mathbf{x}} \left\{ \left\| \mathbf{g}_{\boldsymbol{\theta}}^{U'_s} \right\|_2^2 \right\} + L\gamma_s^2 \sum_{k=1}^K \left\| \bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^{U'_s} - \bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^{U_s} \right\|_2^2 \\
&\leq L\gamma_s^2 \frac{1}{M^2} \mathbb{E}_{\mathbf{x}} \left\{ \sum_{j=0}^{M-1} \left\| \bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^{U_s+j-2(K-k)} \right\|_2^2 \right\} + L\gamma_s^2 \sum_{k=1}^K \left\| \bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^{U'_s} - \bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^{U_s} \right\|_2^2 \\
&\leq \frac{AL}{M} \gamma_s^2 + L\gamma_s^2 \sum_{k=1}^K \left\| \bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^{U'_s} - \bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^{U_s} \right\|_2^2 \\
&= \frac{AL}{M} \gamma_s^2 + L\gamma_s^2 \tilde{P}_1
\end{aligned}$$

where the first inequality follows from $\|\mathbf{x} + \mathbf{y}\|_2^2 \leq 2\|\mathbf{x}\|_2^2 + 2\|\mathbf{y}\|_2^2$. The second inequality is from $\mathbb{E}\{\|\epsilon - \mathbb{E}\{\epsilon\}\|_2^2\} \leq \mathbb{E}\{\|\epsilon\|_2^2\} - \|\mathbb{E}\{\epsilon\}\|_2^2 \leq \mathbb{E}\{\|\epsilon\|_2^2\}$ due to gradient unbiasedness (i.e., $\mathbb{E}_{\mathbf{x}}\{\mathbf{g}_{\boldsymbol{\theta}_{q(k)}}^{U'_s}\} = \bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^{U'_s}$). The last inequality follows from Assumption 4.3.2, and $\tilde{P}_1$

is bounded by

$$\begin{aligned}
\tilde{P}_1 &= \sum_{k=1}^K \left\| \bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^{U'_s} - \bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^{U_s} \right\|_2^2 \\
&\leq \frac{1}{M^2} \sum_{k=1}^K \sum_{j=0}^{M-1} \left\| \bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^{U_s+j-2(K-k)} - \bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^{U_s} \right\|_2^2 \\
&= \frac{L^2}{M^2} \sum_{k=1}^K \sum_{j=0}^{M-1} \left\| \boldsymbol{\theta}_{q(k)}^{U_s} - \boldsymbol{\theta}_{q(k)}^{U_{s-d_{k,j}}} \right\|_2^2 \\
&= \frac{L^2}{M^2} \sum_{k=1}^K \sum_{j=0}^{M-1} \left\| \sum_{\alpha=\max\{0, s-d_{k,j}\}}^{s-1} (\boldsymbol{\theta}_{q(k)}^{U_{\alpha+1}} - \boldsymbol{\theta}_{q(k)}^{U_{\alpha}}) \right\|_2^2 \\
&\leq \frac{L^2}{M^2} \sum_{k=1}^K \sum_{j=0}^{M-1} \sum_{\alpha=\max\{0, s-d_{k,j}\}}^{s-1} \left\| \boldsymbol{\theta}_{q(k)}^{U_{\alpha+1}} - \boldsymbol{\theta}_{q(k)}^{U_{\alpha}} \right\|_2^2 \\
&= \frac{L^2}{M^2} \sum_{k=1}^K \sum_{j=0}^{M-1} \sum_{\alpha=\max\{0, s-d_{k,j}\}}^{s-1} \gamma_{\alpha}^2 \left\| \mathbf{g}_{\boldsymbol{\theta}_{q(k)}}^{U'_s} \right\|_2^2 \\
&\leq \frac{L^2}{M^2} \sum_{k=1}^K \sum_{j=0}^{M-1} \sum_{\alpha=\max\{0, s-d_{k,j}\}}^{s-1} \gamma_{\alpha}^2 \frac{1}{M^2} \sum_{j=0}^{M-1} \left\| \mathbf{g}_{\boldsymbol{\theta}_{q(k)}}^{U_s+j-2(K-k)} \right\|_2^2 \\
&\leq \frac{AL^2}{M^2} \sum_{k=1}^K \sum_{j=0}^{M-1} \sum_{\alpha=\max\{0, s-d_{k,j}\}}^{s-1} \gamma_{\alpha}^2 \leq \gamma_s^2 \frac{AL^2}{M^2} \sum_{k=1}^K \sum_{j=0}^{M-1} (s - \max\{0, s - d_{k,j}\}) \\
&\leq \gamma_s^2 \frac{AL^2}{M^3} \sum_{k=1}^K \sum_{j=0}^{M-1} d_{k,j} \\
&= \gamma_s^2 \frac{AL^2}{M^2} \sum_{k=1}^K \bar{d}_k
\end{aligned}$$

with the first inequality coming from Assumption 4.3.1. On the other hand, the expectation of $\tilde{Q}_2$ is bounded by

$$\begin{aligned}
\mathbb{E}_{\mathbf{x}}\{\tilde{Q}_2\} &= -(\gamma_s - L\gamma_s^2)\mathbb{E}_{\mathbf{x}}\left\{\sum_{k=1}^K(\bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^{U_s})^T(\mathbf{g}_{\boldsymbol{\theta}_{q(k)}}^{U'_s} - \bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^{U_s})\right\} \\
&= -(\gamma_s - L\gamma_s^2)\sum_{k=1}^K(\bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^{U_s})^T(\bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^{U'_s} - \bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^{U_s}) \\
&\leq \frac{\gamma_s - L\gamma_s^2}{2}\sum_{k=1}^K\|\bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^{U_s}\|_2^2 + \frac{\gamma_s - L\gamma_s^2}{2}\tilde{P}_1
\end{aligned}$$

where the second equality follows by the unbiased gradient using SGD, and the inequality comes from $\pm \mathbf{x}^T \mathbf{y} \leq \frac{1}{2}\|\mathbf{x}\|_2^2 + \frac{1}{2}\|\mathbf{y}\|_2^2$.

Taking the expectation of both sides in (A.12) and substituting $\tilde{Q}_1$ and $\tilde{Q}_2$, the inequality is rewritten as

$$\begin{aligned}
\mathbb{E}_{\mathbf{x}}\{f(\boldsymbol{\theta}^{U_{s+1}})\} &\leq f(\boldsymbol{\theta}^{U_s}) - \left(\gamma_s - \frac{L\gamma_s^2}{2}\right)\sum_{k=1}^K\|\bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^{U_s}\|_2^2 + \frac{AL}{M}\gamma_s^2 + L\gamma_s^2\tilde{P}_1 \\
&\quad + \frac{\gamma_s - L\gamma_s^2}{2}\sum_{k=1}^K\|\bar{\mathbf{g}}_{\boldsymbol{\theta}_{q(k)}}^{U_s}\|_2^2 + \frac{\gamma_s - L\gamma_s^2}{2}\tilde{P}_1 \\
&= f(\boldsymbol{\theta}^{U_s}) - \frac{\gamma_s}{2}\|\bar{\mathbf{g}}_{\boldsymbol{\theta}}^{U_s}\|_2^2 + \frac{\gamma_s + L\gamma_s^2}{2}\tilde{P}_1 + \frac{AL}{M}\gamma_s^2 \\
&\leq f(\boldsymbol{\theta}^{U_s}) - \frac{\gamma_s}{2}\|\bar{\mathbf{g}}_{\boldsymbol{\theta}}^{U_s}\|_2^2 + \frac{\gamma_s + L\gamma_s^2}{2}\gamma_s^2\frac{AL^2}{M^2}\sum_{k=1}^K\bar{d}_k + \frac{AL}{M}\gamma_s^2 \\
&= f(\boldsymbol{\theta}^{U_s}) - \frac{\gamma_s}{2}\|\bar{\mathbf{g}}_{\boldsymbol{\theta}}^{U_s}\|_2^2 + \gamma_s^2\left(\frac{AL}{M} + \frac{\gamma_s + L\gamma_s^2}{2}L\frac{AL}{M^2}\sum_{k=1}^K\bar{d}_k\right) \\
&\leq f(\boldsymbol{\theta}^{U_s}) - \frac{\gamma_s}{2}\|\bar{\mathbf{g}}_{\boldsymbol{\theta}}^{U_s}\|_2^2 + \gamma_s^2\frac{AL}{M}\left(1 + \frac{1}{M}\sum_{k=1}^K\bar{d}_k\right) \tag{A.13}
\end{aligned}$$

where the last inequality follows from $L\gamma_s \leq 1$ such that $\frac{\gamma_s + L\gamma_s^2}{2}L = \frac{1}{2}(L\gamma_s + (L\gamma_s)^2) \leq$

1. The proof is now completed. $\square$

A.2.2 Proof of Theorem 4.3.2

Proof. By moving $\frac{\gamma_s}{2} \left\| \bar{\mathbf{g}}_{\boldsymbol{\theta}}^{U_s} \right\|_2^2$ and $\mathbb{E}_{\mathbf{x}}\{f(\boldsymbol{\theta}^{U_{s+1}})\}$ to the LHS and the RHS of (4.15) respectively, and multiplying both sides by 2, we have

$$\gamma_s \left\| \bar{\mathbf{g}}_{\boldsymbol{\theta}}^{U_s} \right\|_2^2 \leq 2(f(\boldsymbol{\theta}^{U_s}) - \mathbb{E}_{\mathbf{x}}\{f(\boldsymbol{\theta}^{U_{s+1}})\}) + 2\gamma_s^2 \frac{AL}{M} \left(1 + \frac{1}{M} \sum_{k=1}^K \bar{d}_k\right). \quad (\text{A.14})$$

Take full expectation on both sides of (A.14), and it leads to

$$\gamma_s \mathbb{E}\left\{ \left\| \bar{\mathbf{g}}_{\boldsymbol{\theta}}^{U_s} \right\|_2^2 \right\} \leq 2(\mathbb{E}\{f(\boldsymbol{\theta}^{U_s})\} - \mathbb{E}\{f(\boldsymbol{\theta}^{U_{s+1}})\}) + 2\gamma_s^2 \frac{AL}{M} \left(1 + \frac{1}{M} \sum_{k=1}^K \bar{d}_k\right). \quad (\text{A.15})$$

By summing both sides of (A.15) from 0 to $S-1$, and dividing it by $\mathbb{T}_S = \sum_{s=0}^{S-1} \gamma_s$, it becomes

$$\begin{aligned} \frac{1}{\mathbb{T}_S} \sum_{s=0}^{S-1} \gamma_s \mathbb{E}\left\{ \left\| \bar{\mathbf{g}}_{\boldsymbol{\theta}}^{U_s} \right\|_2^2 \right\} &\leq \frac{2(f(\boldsymbol{\theta}^0) - \mathbb{E}\{f(\boldsymbol{\theta}^{U_S})\})}{\mathbb{T}_S} + \frac{2\frac{AL}{M} \left(1 + \frac{1}{M} \sum_{k=1}^K \bar{d}_k\right) \sum_{s=0}^{S-1} \gamma_s^2}{\mathbb{T}_S} \\ &\leq \frac{2(f(\boldsymbol{\theta}^0) - f(\boldsymbol{\theta}^*))}{\mathbb{T}_S} + \frac{2\frac{AL}{M} \left(1 + \frac{1}{M} \sum_{k=1}^K \bar{d}_k\right) \sum_{s=0}^{S-1} \gamma_s^2}{\mathbb{T}_S}. \end{aligned}$$

where the last inequality comes from $f(\boldsymbol{\theta}^*) \leq \mathbb{E}\{f(\boldsymbol{\theta}^{U_S})\}$. $\square$

A.2.3 Proof of Theorem 4.3.3

Proof. The proof is started from (A.15) as the constant learning rate is a special case in Theorem 4.3.2. By setting $\gamma_s = \gamma$, Eq. (A.15) is rewritten as

$$\mathbb{E}\left\{ \left\| \bar{\mathbf{g}}_{\boldsymbol{\theta}}^{U_s} \right\|_2^2 \right\} \leq \frac{2(\mathbb{E}\{f(\boldsymbol{\theta}^{U_s})\} - \mathbb{E}\{f(\boldsymbol{\theta}^{U_{s+1}})\})}{\gamma} + 2\gamma \frac{AL}{M} \left(1 + \frac{1}{M} \sum_{k=1}^K \bar{d}_k\right). \quad (\text{A.16})$$

Summing both sides of (A.16) from $s = 0$ to $S - 1$ and dividing them by S , it leads to

$$\begin{aligned} \frac{1}{S} \sum_{s=0}^{S-1} \mathbb{E}\{\|\bar{\mathbf{g}}_{\boldsymbol{\theta}}^{U_s}\|_2^2\} &\leq \frac{2(f(\boldsymbol{\theta}^0) - \mathbb{E}\{f(\boldsymbol{\theta}^{U_s})\})}{\gamma S} + 2\gamma \frac{AL}{M} \left(1 + \frac{1}{M} \sum_{k=1}^K \bar{d}_k\right) \\ &\leq \frac{2(f(\boldsymbol{\theta}^0) - f(\boldsymbol{\theta}^*))}{\gamma S} + 2\gamma \frac{AL}{M} \left(1 + \frac{1}{M} \sum_{k=1}^K \bar{d}_k\right). \end{aligned} \quad (\text{A.17})$$

Substituting $\gamma = \epsilon \sqrt{M(f(\boldsymbol{\theta}^0) - f(\boldsymbol{\theta}^*)) / (SAL(1 + (1/M) \sum_{k=1}^K \bar{d}_k))}$ into (A.17), the RHS becomes

$$\begin{aligned} &\frac{2(f(\boldsymbol{\theta}^0) - f(\boldsymbol{\theta}^*)) + 2\gamma^2 SAL \left(1 + (1/M) \sum_{k=1}^K \bar{d}_k\right) / M}{\gamma S} \\ &= \frac{(2 + 2\epsilon^2)(f(\boldsymbol{\theta}^0) - f(\boldsymbol{\theta}^*))}{S\epsilon \sqrt{M(f(\boldsymbol{\theta}^0) - f(\boldsymbol{\theta}^*)) / (SAL(1 + (1/M) \sum_{k=1}^K \bar{d}_k))}} \\ &= \frac{(2 + 2\epsilon^2)}{\epsilon} \sqrt{AL(f(\boldsymbol{\theta}^0) - f(\boldsymbol{\theta}^*)) \left(1 + (1/M) \sum_{k=1}^K \bar{d}_k\right) / (MS)}. \end{aligned}$$

Since the LHS of (A.17) is the average of $\mathbb{E}\{\|\bar{\mathbf{g}}_{\boldsymbol{\theta}}^{U_s}\|_2^2\}$ for $s = 0, 1, \dots, S - 1$, we have

$$\min_{s \in \{0, 1, \dots, S-1\}} \mathbb{E}\{\|\bar{\mathbf{g}}_{\boldsymbol{\theta}}^{U_s}\|_2^2\} \leq \frac{(2 + 2\epsilon^2)}{\epsilon} \sqrt{AL(f(\boldsymbol{\theta}^0) - f(\boldsymbol{\theta}^*)) \left(1 + (1/M) \sum_{k=1}^K \bar{d}_k\right) / (MS)}$$

which completes the proof. $\square$

A.3 Appendix for Chapter 6

A.3.1 Proof for Theorem 6.2.1

Proof. According to Lemma 6.2.2, the MP inverse of $\mathbf{H}_t$ can be calculated using $\mathbf{H}_{t-1}^\dagger$ as follows:

$$\mathbf{H}_t^\dagger = \begin{bmatrix} \mathbf{H}_{t-1}^\dagger - \mathbf{C}_t^\dagger \mathcal{H}_t \mathbf{H}_{t-1}^\dagger - \mathbf{V}_t \mathcal{H}_t \mathbf{H}_{t-1}^\dagger & \mathbf{C}_t^\dagger + \mathbf{V}_t \end{bmatrix} \quad (\text{A.18})$$

where

$$\begin{aligned} \mathbf{C}_t &= \mathcal{H}_t(\mathbf{I} - \mathbf{H}_{t-1}^\dagger \mathbf{H}_{t-1}) \\ \mathbf{K}_t &= (\mathbf{I} + (\mathbf{I} - \mathbf{C}_t \mathbf{C}_t^\dagger) \mathcal{H}_t \mathbf{H}_{t-1}^\dagger \mathbf{H}_{t-1}^{\dagger T} \mathcal{H}_t^T (\mathbf{I} - \mathbf{C}_t \mathbf{C}_t^\dagger))^{-1} \\ \mathbf{V}_t &= (\mathbf{I} - \mathbf{C}_t^\dagger \mathcal{H}_t) \mathbf{H}_{t-1}^\dagger \mathbf{H}_{t-1}^{\dagger T} \mathcal{H}_t^T \mathbf{K}_t (\mathbf{I} - \mathbf{C}_t \mathbf{C}_t^\dagger). \end{aligned}$$

Since

$$\hat{\mathbf{w}}^{(t)} = \mathbf{H}_t^\dagger \mathbf{z}_t = \begin{bmatrix} \mathbf{H}_{t-1}^\dagger \\ \mathcal{H}_t \end{bmatrix}^\dagger \begin{bmatrix} \mathbf{z}_{t-1} \\ \mathbf{z}_t \end{bmatrix},$$

by replacing $\mathbf{H}_t^\dagger$ with (A.18), we obtain

$$\begin{aligned} \hat{\mathbf{w}}^{(t)} &= \begin{bmatrix} \mathbf{H}_{t-1}^\dagger - \mathbf{C}_t^\dagger \mathcal{H}_t \mathbf{H}_{t-1}^\dagger - \mathbf{V}_t \mathcal{H}_t \mathbf{H}_{t-1}^\dagger & \mathbf{C}_t^\dagger + \mathbf{V}_t \end{bmatrix} \begin{bmatrix} \mathbf{z}_{t-1} \\ \mathbf{z}_t \end{bmatrix} \\ &= \mathbf{H}_{t-1}^\dagger \mathbf{z}_{t-1} - \mathbf{C}_t^\dagger \mathcal{H}_t \mathbf{H}_{t-1}^\dagger \mathbf{z}_{t-1} - \mathbf{V}_t \mathcal{H}_t \mathbf{H}_{t-1}^\dagger \mathbf{z}_{t-1} + (\mathbf{C}_t^\dagger + \mathbf{V}_t) \mathbf{z}_t \\ &= \hat{\mathbf{w}}^{(t-1)} - (\mathbf{C}_t^\dagger + \mathbf{V}_t) \mathcal{H}_t \hat{\mathbf{w}}^{(t-1)} + (\mathbf{C}_t^\dagger + \mathbf{V}_t) \mathbf{z}_t \\ &= \hat{\mathbf{w}}^{(t-1)} + \mathbf{G}_t (\mathbf{z}_t - \mathcal{H}_t \hat{\mathbf{w}}^{(t-1)}) \end{aligned}$$

where $\mathbf{G}_t = \mathbf{C}_t^\dagger + \mathbf{V}_t$. □

A.3.2 Proof for Lemma 6.2.3

Proof. The lemma can be proved in the similar way of matrix manipulations as in Theorem 6.2.1. Having obtained $\mathbf{H}_t^\dagger$ in (A.18), $\mathbf{Q}_t$ can be rewritten as

$$\begin{aligned}
\mathbf{Q}_t &= \mathbf{I} - \begin{bmatrix} \mathbf{H}_{t-1} \\ \mathbf{H}_t \end{bmatrix}^\dagger \begin{bmatrix} \mathbf{H}_{t-1} \\ \mathbf{H}_t \end{bmatrix} \\
&= \mathbf{I} - \begin{bmatrix} \mathbf{H}_{t-1}^\dagger - \mathbf{C}_t^\dagger \mathbf{H}_t \mathbf{H}_{t-1}^\dagger - \mathbf{V}_t \mathbf{H}_t \mathbf{H}_{t-1}^\dagger & \mathbf{C}_t^\dagger + \mathbf{V}_t \end{bmatrix} \begin{bmatrix} \mathbf{H}_{t-1} \\ \mathbf{H}_t \end{bmatrix} \\
&= \mathbf{I} - \mathbf{H}_{t-1}^\dagger \mathbf{H}_{t-1} - \mathbf{C}_t^\dagger \mathbf{H}_t \mathbf{H}_{t-1}^\dagger \mathbf{H}_{t-1} - \mathbf{V}_t \mathbf{H}_t \mathbf{H}_{t-1}^\dagger \mathbf{H}_{t-1} + (\mathbf{C}_t^\dagger + \mathbf{V}_t) \mathbf{H}_t \\
&= \mathbf{Q}_{t-1} - (\mathbf{C}_t^\dagger + \mathbf{V}_t)(\mathbf{H}_t - \mathbf{H}_t \mathbf{H}_{t-1}^\dagger \mathbf{H}_{t-1}) \\
&= \mathbf{Q}_{t-1} - (\mathbf{C}_t^\dagger + \mathbf{V}_t) \mathbf{H}_t (\mathbf{I} - \mathbf{H}_{t-1}^\dagger \mathbf{H}_{t-1}) \\
&= \mathbf{Q}_{t-1} - \mathbf{G}_t \mathbf{H}_t \mathbf{Q}_{t-1}
\end{aligned}$$

which completes the proof. □

A.3.3 Proof for Lemma 6.2.4

Proof. The lemma can be proved using similar techniques adopted in proving Theorem 6.2.1:

$$\begin{aligned}
\mathbf{R}_t &= \begin{bmatrix} \mathbf{H}_{t-1} \\ \mathbf{H}_t \end{bmatrix}^\dagger \begin{bmatrix} \mathbf{H}_{t-1} \\ \mathbf{H}_t \end{bmatrix}^{\dagger T} \\
&= \begin{bmatrix} (\mathbf{I} - \mathbf{C}_t^\dagger \mathbf{H}_t - \mathbf{V}_t \mathbf{H}_t) \mathbf{H}_{t-1}^\dagger & \mathbf{C}_t^\dagger + \mathbf{V}_t \end{bmatrix} \begin{bmatrix} \mathbf{H}_{t-1}^{\dagger T} (\mathbf{I} - \mathbf{C}_t^\dagger \mathbf{H}_t - \mathbf{V}_t \mathbf{H}_t)^T \\ (\mathbf{C}_t^\dagger + \mathbf{V}_t)^T \end{bmatrix} \\
&= (\mathbf{I} - \mathbf{C}_t^\dagger \mathbf{H}_t - \mathbf{V}_t \mathbf{H}_t) \mathbf{R}_{t-1} (\mathbf{I} - \mathbf{C}_t^\dagger \mathbf{H}_t - \mathbf{V}_t \mathbf{H}_t)^T + (\mathbf{C}_t^\dagger + \mathbf{V}_t) (\mathbf{C}_t^\dagger + \mathbf{V}_t)^T.
\end{aligned}$$

Since $\mathbf{G}_t = \mathbf{C}_t^\dagger + \mathbf{V}_t$, we have

$$\mathbf{R}_t = (\mathbf{I} - \mathbf{G}_t \mathbf{H}_t) \mathbf{R}_{t-1} (\mathbf{I} - \mathbf{G}_t \mathbf{H}_t)^T + \mathbf{G}_t \mathbf{G}_t^T.$$

□

A.3.4 Proof for Theorem 6.2.2

Proof. The proof for the update rule (6.20)-(6.26) can be immediately obtained by combining Theorem 6.2.1, Lemmas 6.2.3 and 6.2.4.

To validate the initialization, a recursion using (6.20)-(6.26) can be simply performed after fetching the first data block $\mathbf{H}_1$ and $\mathbf{z}_1$. In such case, $\hat{\mathbf{w}}^{(1)}$, $\mathbf{Q}_1$ and $\mathbf{R}_1$

are updated as follows:

$$\mathbf{C}_1 = \mathbf{H}_1 \mathbf{Q}_0 = \mathbf{H}_1$$

$$\mathbf{K}_1 = (\mathbf{I} + (\mathbf{I} - \mathbf{C}_1 \mathbf{C}_1^\dagger) \mathbf{H}_1 \mathbf{R}_0 \mathbf{H}_1^T (\mathbf{I} - \mathbf{C}_1 \mathbf{C}_1^\dagger))^{-1} = \mathbf{I}$$

$$\mathbf{V}_1 = (\mathbf{I} - \mathbf{C}_1^\dagger \mathbf{H}_1) \mathbf{R}_0 \mathbf{H}_1^T \mathbf{K}_1 (\mathbf{I} - \mathbf{C}_1 \mathbf{C}_1^\dagger) = \mathbf{0}$$

$$\mathbf{G}_1 = \mathbf{C}_1^\dagger + \mathbf{V}_1 = \mathbf{H}_1^\dagger$$

$$\hat{\mathbf{w}}^{(1)} = \hat{\mathbf{w}}^{(0)} + \mathbf{G}_1 (\mathbf{z}_1 - \mathbf{H}_1 \hat{\mathbf{w}}^{(0)}) = \mathbf{H}_1^\dagger \mathbf{z}_1 = \mathbf{H}_1^\dagger \mathbf{z}_1 \quad (\text{A.19})$$

$$\mathbf{Q}_1 = \mathbf{Q}_0 - \mathbf{G}_1 \mathbf{H}_1 \mathbf{Q}_0 = \mathbf{I} - \mathbf{H}_1^\dagger \mathbf{H}_1 \quad (\text{A.20})$$

$$\begin{aligned} \mathbf{R}_1 &= (\mathbf{I} - \mathbf{G}_1 \mathbf{H}_1) \mathbf{R}_0 (\mathbf{I} - \mathbf{G}_1 \mathbf{H}_1)^T + \mathbf{G}_1 \mathbf{G}_1^T \\ &= \mathbf{H}_1^\dagger \mathbf{H}_1^{\dagger T} = \mathbf{H}_1^\dagger \mathbf{H}_1^{\dagger T}. \end{aligned} \quad (\text{A.21})$$

It is observed that $\hat{\mathbf{w}}^{(1)}$, $\mathbf{Q}_1$ and $\mathbf{R}_1$ given by (A.19), (A.20) and (A.21) are identical to (6.6), (6.17) and (6.19) at $t = 1$. This indicates that the initialization (6.27) is a proper way to start the recursion. $\square$

A.3.5 Proof for Lemma 6.2.5

Proof. 1) *sufficiency*: If $\mathbf{H}_t$ lies in $\mathbf{H}_{t-1}$'s row space, there exists a weight matrix $\mathbf{W}$ such that $\mathbf{H}_t = \mathbf{W} \mathbf{H}_{t-1}$. Therefore,

$$\begin{aligned} \mathbf{C}_t &= \mathbf{H}_t (\mathbf{I} - \mathbf{H}_{t-1}^\dagger \mathbf{H}_{t-1}) \\ &= \mathbf{W} (\mathbf{H}_{t-1} - \mathbf{H}_{t-1} \mathbf{H}_{t-1}^\dagger \mathbf{H}_{t-1}). \end{aligned}$$

For any matrix $\mathbf{H}_{t-1}$, according to Definition 6.1.1, we have $\mathbf{H}_{t-1} = \mathbf{H}_{t-1} \mathbf{H}_{t-1}^\dagger \mathbf{H}_{t-1}$. This leads to $\mathbf{C}_t = \mathbf{0}$.

2) *necessity*: If $\mathbf{C}_t = \mathbf{0}$, according to (6.12), we have

$$\begin{aligned}\mathbf{C}_t &= \mathcal{H}_t(\mathbf{I} - \mathbf{H}_{t-1}^\dagger \mathbf{H}_{t-1}) = \mathbf{0} \\ \Rightarrow \mathcal{H}_t &= \mathcal{H}_t \mathbf{H}_{t-1}^\dagger \mathbf{H}_{t-1}\end{aligned}$$

which reveals that $\mathcal{H}_t$ can be linearly combined by the rows of $\mathbf{H}_{t-1}$ with a weighting matrix $\mathcal{H}_t \mathbf{H}_{t-1}^\dagger$. That is, $\mathcal{H}_t$ lies in the row space of matrix $\mathbf{H}_{t-1}$. $\square$

A.3.6 Proof for Lemma 6.2.6

Proof. 1) *sufficiency*: If the accumulated data $\mathbf{H}_{t-1}$ is of full column rank, we have $\mathbf{H}_{t-1}^\dagger \mathbf{H}_{t-1} = \mathbf{I}$ which indicates

$$\mathbf{Q}_{t-1} = \mathbf{I} - \mathbf{H}_{t-1}^\dagger \mathbf{H}_{t-1} = \mathbf{0}.$$

2) *necessity*: If $\mathbf{Q}_{t-1} = \mathbf{0}$, we have

$$\begin{aligned}\mathbf{I} &= \mathbf{H}_{t-1}^\dagger \mathbf{H}_{t-1} \\ &= (\mathbf{H}_{t-1}^T \mathbf{H}_{t-1})^\dagger \mathbf{H}_{t-1}^T \mathbf{H}_{t-1}\end{aligned}$$

which indicates that $\mathbf{H}_{t-1}^T \mathbf{H}_{t-1}$ is a full-rank matrix. Hence, $\mathbf{H}_{t-1}$ has full column rank. $\square$

A.3.7 Proof for Corollary 6.2.2

Proof. If $\mathbf{Q}_{t-1} = \mathbf{0}$, it subsequently leads to

$$\mathbf{Q}_t = \mathbf{Q}_{t-1} - \mathbf{G}_t \mathbf{H}_t \mathbf{Q}_{t-1} = \mathbf{0}.$$

Therefore, both $\mathbf{Q}_{t-1}$ and $\mathbf{Q}_t$ can be eliminated. Since $\mathbf{Q}_{t-1}$ also leads to $\mathbf{C}_t = \mathbf{0}$, according to Corollary 6.2.1, $\mathbf{G}_t$, $\mathbf{K}_t$ and $\mathbf{R}_t$ are reduced to (6.29), (6.30) and (6.32).

By substituting $\mathbf{G}_t$ with (6.29), $\mathbf{R}_t$ in (6.32) can be further rewritten as

$$\begin{aligned} \mathbf{R}_t &= \mathbf{R}_{t-1} - \mathbf{G}_t \mathbf{H}_t \mathbf{R}_{t-1} - \mathbf{R}_{t-1} \mathbf{H}_t^T \mathbf{G}_t^T + \mathbf{G}_t \mathbf{K}_t^{-1} \mathbf{G}_t^T. \\ &= \mathbf{R}_{t-1} - \mathbf{R}_{t-1} \mathbf{H}_t^T \mathbf{K}_t \mathbf{H}_t \mathbf{R}_{t-1} \\ &= \mathbf{R}_{t-1} - \mathbf{R}_{t-1} \mathbf{H}_t^T (\mathbf{I} + \mathbf{H}_t \mathbf{R}_{t-1} \mathbf{H}_t^T)^{-1} \mathbf{H}_t \mathbf{R}_{t-1} \end{aligned}$$

where the second equation makes use of $\mathbf{G}_t \mathbf{K}_t^{-1} = \mathbf{R}_{t-1} \mathbf{H}_t^T$. Subsequently, since

$$\mathbf{I} = \mathbf{K}_t \mathbf{K}_t^{-1} = \mathbf{K}_t (\mathbf{I} + \mathbf{H}_t \mathbf{R}_{t-1} \mathbf{H}_t^T)$$

we have

$$\mathbf{K}_t = \mathbf{I} - \mathbf{K}_t \mathbf{H}_t \mathbf{R}_{t-1} \mathbf{H}_t^T.$$

This expands (6.29) into

$$\begin{aligned} \mathbf{G}_t &= \mathbf{R}_{t-1} \mathbf{H}_t^T (\mathbf{I} - \mathbf{K}_t \mathbf{H}_t \mathbf{R}_{t-1} \mathbf{H}_t^T) \\ &= (\mathbf{R}_{t-1} - \mathbf{R}_{t-1} \mathbf{H}_t^T \mathbf{K}_t \mathbf{H}_t \mathbf{R}_{t-1}) \mathbf{H}_t^T = \mathbf{R}_t \mathbf{H}_t^T. \end{aligned}$$

□